

ISSN 0933-9434 | Mai/Juni 1989 | 3. Jg./Heft 3

öS 150,- | DM 19,80 | sfr 19,80

MS OS/2:

Forth, Lisp, Modula-2 und Basic unter OS/2

Anwendungsentwicklung unter OS/2

Debuggen von Multithread-Programmen

C-Know-how:

Die Gültigkeitsbereiche von Bezeichnern

SAA-Serie:

Pull-Down-Menüverwaltung

Windows:

Eigene Dialogfenster-Klassen

Brandneu: Microsoft QuickPascal

Ausgabe Mai/Juni 1989





MS OS/2

Forth, Lisp, Modula-2 und Basic unter OS/2 4

OS/2-Programmierer müssen sich nicht auf eine Sprache beschränken. Eine große Anzahl von Compilern und Interpretern ist für den geschützten Modus von OS/2 verfügbar. Dieser Artikel untersucht vier von ihnen – Forth, Lisp, Modula-2 und Basic. Jede dieser Sprachen hat ihre eigenen Vorteile.

Anwendungsentwicklung unter OS/2 28

OS/2 bietet erhebliche Fortschritte in der Produktivität, egal ob Sie Programme für OS/2 oder für DOS entwickeln, aber es erfordert auch, daß Sie einige Programmierannahmen überdenken, die Sie unter DOS gemacht haben. Dieser Artikel hilft beim Einstieg.

Debuggen von Multithread-Programmen 38

CodeView in der Version 2.2 bietet einige neue und verbesserte Kommandos, um die Fehlersuche in Programmen mit mehreren Threads zu unterstützen. Wir zeigen, wie es geht.

Pascal

Microsoft kündigt QuickPascal an 58

Microsoft hat mit der Vorstellung von Microsoft QuickPascal den Pascal-Massenmarkt betreten. Es ist ein Produkt, das durch Geschwindigkeit, umfangreiche Fähigkeiten und die Unterstützung erweiterter Programmierkonstruktionen neue Maßstäbe setzt.

C

Pull-Down-Menüverwaltung 60

Diese Folge über eine SAA-Benutzeroberfläche in C ist einem übergeordneten Modul, der Pull-Down-Menüverwaltung, gewidmet. Unserem Ziel, der Erstellung SAA- bzw. DOS-4.0-konsistenter Benutzeroberflächen, kommen wir damit sehr viel näher.

Die Gültigkeitsbereiche von Bezeichnern 83

Es werden die verschiedenen Konzepte des Gebrauchs von Bezeichnern sowohl innerhalb einer Quellcodedatei, als auch über verschiedene hinweg erörtert. Man kennt C nicht richtig, wenn man nicht alle Attribute, die ein Bezeichner haben kann, richtig versteht. Das Erlernen dieser Bezeichner führt zu einer besseren und professionelleren Programmierung.

Microsoft Windows

Eigene Dialogfenster-Klassen 93

Nachdem sich der erste Beitrag über Fenster und Fensterklassen unter Windows im Microsoft System Journal 3/4 '89 mit Unterklassen befaßt hat, steht diesmal die Erstellung eigener Fensterklassen im Vordergrund. Mit diesen lassen sich eigenständige Dialogobjekte entwickeln und vielseitig anwenden.

Rubriken

Termine .... 47 — Die Termine des Microsoft-Instituts

Mitteilungen .... 49 — Neue Produkte, Aktuelles

Buchbesprechung .... 92 — Die Quick-C-Toolbox

Impressum 110

Inserentenverzeichnis 56

Programmierung im geschützten Modus von OS/2:

Forth, Lisp, Modula-2 und Basic unter OS/2

Wenn Sie auf einer einsamen Insel stranden würden, welche Programmiersprache hätten Sie dann gern ins Rettungsboot mitgenommen? Wenn ich in dieser Lage wäre, dann fiele die Wahl ohne Zweifel auf C. Glücklicherweise müssen wir aber keine solche Wahl treffen. OS/2-Programmierer müssen sich nicht wie auf einer einsamen Insel vorkommen, und Ihre Arbeit ist erst recht nicht auf eine Sprache beschränkt. Eine große Anzahl von Compilern und Interpretern ist für den geschützten Modus von OS/2 verfügbar.

Dieser Artikel untersucht vier von ihnen - Forth, Lisp, Modula-2 und Basic. Jede Sprache hat ihre eigenen Vorteile, und mit jeder kann man Dinge erledigen, die in C nicht unmittelbar möglich sind.

UR/Forth von Laboratory Microsystems Inc (LMI) ist für den geschützten Modus von OS/2, für 386-Computer, für MS-DOS und Unix-Computer unter dem 68000er verfügbar. Das in diesem Artikel verwendete Modula-2-System für OS/2 ist von Stony Brook Software. Hiervon ist auch eine MS-DOS-Version verfügbar. Die Version 6.0 von Microsoft Basic erzeugt Programme für MS-DOS und OS/2. Der hier vorgestellte Lisp-Interpreter ist eine erweiterte Version der von David Betz frei vertriebenen XLisp-Version, die ich auf den geschützten Modus von OS/2 übertragen habe. Sie ist in verschiedenen Mailboxen erhältlich. Es sind aber auch noch andere Sprachen verfügbar, wie Cobol und Fortran. Die in diesem Artikel bezogene Position entspricht jedoch der aus *The Tao of Programming*, einem älteren Text, der von Geoffrey James entdeckt und übersetzt wurde. Dort steht:

»Der Assembler veranlaßte die Geburt des Compilers. Inzwischen gibt es zehntausend Sprachen. Jede Sprache verfolgt ihren eigenen Zweck. Jede Sprache drückt das Hin und Her der Software aus. Jede Sprache hat seinen Platz im Tao. Programmieren Sie aber nicht in Cobol, wenn es zu vermeiden ist.« (Geoffrey James, The Tao of Programming, InfoBooks, 1987, Seite 7).

Wir werden uns bei jedem Produkt mit den folgenden Themen beschäftigen:

- Wie kann der Anwender die Betriebssystem-Schnittstelle (API) aufrufen? Wie gut sind OS/2-Aufrufe in die Sprache integriert?
- Welche Umwandlungen sind zwischen den OS/2- und den spracheigenen Datentypen nötig? Wie schwierig ist das Erzeugen von OS/2-Datenstrukturen?
- Im Gegensatz zu C haben einige Sprachen Fähigkeiten für Multitasking. Beim Portieren auf OS/2 sollten diese Prozeß-Fähigkeiten mit Threads implementiert sein. Wie läßt sich in OS/2 mit diesen Fähigkeiten arbeiten?
- Wie unterscheidet sich dieses Produkt von MS-DOS- oder 386-Implementierungen desselben Herstellers? Welche OS/2-Fähigkeiten nützt dieses Programm aus?

Nebenbei werden wir auch feststellen, daß einige Produkte nützliche Zusätze bieten, wie etwa einen eingebauten Assembler, oder andere Formen der Programmierung in verschiedenen Sprachen, auch OS/2-Grafik (aber nicht unter dem Presentation Manager), Zugriff auf Ein-/Ausgabe-Ports und eingebaute Fehlersuchunterstützung.

Ein OS/2 Rosetta-Stein

Im Jahre 196 vor Christi Geburt, unter der Herrschaft von Ptolemäus V., übersetzte ein Schreiber ein Dekret in drei verschiedene Sprachen: Griechisch, Ägyptisch und Hieroglyphen. Dieser Fund, der als Rosetta-Stein bekannt ist, ermöglichte im frühen neunzehnten Jahrhundert die Entzifferung der ägyptischen Hieroglyphen.

In derselben Art und Weise zeigt das Programm in *Abbildung 1* ein ansonsten unwichtiges Programm in C, Forth, Lisp, Modula-2 und Basic. Die Absicht dabei ist, die sonst kryptischen Programme in anderen Sprachen einem C-Programmierer verständlich zu machen.

Das Programm bearbeitet in einer Schleife alle vorhandenen Modul-Handles und gibt alle Namen der gerade geladenen ausführbaren dynamischen Linkbibliotheken in allen Sitzungen aus. Das Programm führt 65535 Aufrufe von DosGetModName aus, und braucht unabhängig von der verwendeten Sprache geraume Zeit (DosGetModName ist wahrscheinlich die langsamste Funktion von OS/2).

Abbildung 1 zeigt auch eine Beispielausgabe des Programms. Sie können sehen, daß ich OS/2 vom Diskettenlaufwerk A gestartet habe und einen Kommandozeileneditor verwende (ALIAS.DLL). Ebenso sind der Epsilon-Editor von Lugaru Software, UR/Forth und OS2XLisp gestartet.

Jedes der Programme beschreitet seinen eigenen Weg bei den OS/2-Funktionsaufrufen. Der Aufruf von OS/2 unter Microsoft C erfordert die Verwendung einer Include-Datei, die die Funktionsprototypen enthält. Eine ähnliche Technik wird auch in Modula-2 und Basic verwendet. Die Art, wie externe Funktionen deklariert werden, ist aber unterschiedlich. Da C nicht die Pascal-Aufrufkonventionen verwendet, bedarf es des Schlüsselworts APIENTRY (far pascal) für die OS/2-Funktionen. Sowohl Modula-2 als auch Basic verwenden bereits die Pascal-Aufrufkonventionen. Deshalb benötigen sie keine besondere Behandlung bei OS/2-Funktionsaufrufen.

UR/Forth und OS2XLisp sind beide interaktive Programmiersysteme und keine batchorientierten Compiler. Beide können keine Include-Dateien verarbeiten. In UR/Forth besitzt nahezu jeder OS/2-API-Aufruf einen eingebauten Header. Nur einige Funktionen (wie KbdRegister und VioRegister), die von Forth nicht richtig genutzt werden können, sind ausgenommen.

In C (Microsoft C 5.1):

```
/* from bsedos.h */
USHORT APIENTRY DosGetModName(HMODULE, USHORT, PCHAR);

/* enumdll.c */
#include <stdio.h>

#define INCL_DOSMODULEMGR
#include "os2.h"

main()
{
    char buf[128];
    register unsigned int i;

    for (i=0; i<=0xFFFF; i++)
        if (! DosGetModName(i, 128, buf))
            printf("%u\t%s\n", i, buf);
}
```

In Forth (Laboratory Microsystems 80286 UR/Forth 1.1):

```
\ no header files

CREATE BUF 128 ALLOT                    \ create a buffer

: ENUMDLL                               \ define a new word
   65535 0 DO                           \ for i=0 to 65535 do
      I 128 DS0 BUF DOSGETMODNAME
                                 \ DosGetModName(i,128,ds:buf)
      0= IF                             \ if no error
          CR                            \ carriage return
          I 5 U.R 3 SPACES              \ display i nicely
          BUF -ASCIIZ COUNT TYPE        \ convert ASCII string
                                        \ and print it
      THEN
   LOOP ;

ENUMDLL                                 \ invoke the word
```

In Lisp (OS2XLISP Version 1.10):

```
; no header files

(define doscalls (loadmodule "DOSCALLS"))
(define dosgetmodname (getprocaddr doscalls "DOSGETMODNAME"))
(define buf (make-string 32 128))           ; string of 128
spaces

(dotimes
    (i #xFFFF)
    (if (zerop (call dosgetmodname (word i) (word 128) buf))
    ; then
        (format stdout "~A\t~A\n" i buf)))
```

In Modula-2 (Stony Brook Software Modula-2 für OS/2):

```
(* from doscalls.def *)
PROCEDURE DosGetModName ['DOSGETMODNAME'] (
ModuleHandle:CARDINAL; (* the module handle to get name for *)
BufferLength:CARDINAL; (* the length of the output buffer *)
Buffer      : Asciiz   (* the address of output buffer *)
) : CARDINAL;

(* enumdll.mod *)
MODULE ENUMDLL;

FROM SYSTEM IMPORT ADR;

FROM InOut IMPORT Write, WriteLn, WriteString, WriteInt;

IMPORT DosCalls;

VAR
    i : CARDINAL;
    buf : ARRAY [0..128] OF CHAR;

BEGIN
    FOR i := 0 TO 65535 DO
      IF DosCalls.DosGetModName(i, 128, ADR(buf)) = 0 THEN
        WriteInt(i, 5); Write(CHAR(9)); WriteString(buf);
        WriteLn;
        END
    END;

END ENUMDLL.
```

In BASIC (Microsoft BASIC Compiler 6.0):

```
' enumdll.bas
' compile with /d so you can Ctrl-Break

' from bsedospe.bi
DECLARE FUNCTION DosGetModName%( _
        BYVAL P1 AS INTEGER,_
        BYVAL P2 AS INTEGER,_
        BYVAL P3s AS INTEGER,_
        BYVAL P3o AS INTEGER)

' include file for Program Execution Support
REM $INCLUDE: 'bsedospe.bi'

buf$ = SPACE$(128)

FOR i% = 0 TO 32765                     ' only signed integers
available
   Result = DosGetModName%(i%, 128, VARSEG(buf$), SADD(buf$))
   IF Result = 0 THEN
      PRINT i%, RTRIM$(buf$)
      buf$ = SPACE$(128)         ' have to reset buffer,
because
   END IF                 ' BASIC doesn't know about ASCIIZ
NEXT i%
```

Ausgabe:

```
   140      A:\HARDERR.EXE
   220      D:\OS2\DLL\BMSCALLS.DLL
   380      D:\OS2\SYS\SHELL.EXE
   630      A:\SWAPPER.EXE
   750      D:\OS2\DLL\BKSCALLS.DLL
   930      D:\OS2\DLL\ANSICALL.DLL
  1210      D:\OS2\DLL\EPS-LIB.DLL
  1230      D:\OS2\DLL\MOUCALLS.DLL
  1240      D:\OS2\DLL\QUECALLS.DLL
  1330      D:\OS2\DLL\SESMGR.DLL
  1340      D:\OS2\DLL\BVSCALLS.DLL
  1380      D:\OS2\DLL\VIOCALLS.DLL
  1390      D:\OS2\DLL\KBDCALLS.DLL
  1480      D:\OS2\DLL\DOSCALL1.DLL
  1490      D:\OS2\DLL\NLS.DLL
  1550      D:\OS2\DLL\MSG.DLL
  1590      D:\OS2\E\EPSILON.EXE
  2240      D:\OS2\DLL\MONCALLS.DLL
  2320      D:\URFOS2\FORTH.EXE
  2960      D:\OS2\DLL\CRTLIB.DLL
  2980      E:\XLISP\NEW\OS2XLISP.EXE
  3190      D:\OS2\DLL\ALIAS.DLL
  3630      D:\OS2\SYS\CMD.EXE
```

Abbildung 1: *Ein OS/2-Rosetta-Stein.*

```
FORGET ENUMDLL      \ get rid of previous definition
CREATE BUF 128 ALLOT
: MODULE? ( n -- flag )   128 DSØ BUF DOSGETMODNAME Ø= ;
: .ASCIIZ ( addr -- )   -ASCIIZ COUNT TYPE ;
: .MODULE ( n -- )   CR   5 U.R   3 SPACES   BUF .ASCIIZ ;
: ENUMDLL ( -- )   65535 Ø DO I MODULE? IF I .MODULE THEN LOOP
;
```

Listing 1: *Eine verbesserte Implementierung des Rosetta-Stein-Programms in Forth.*

OS2XLisp verwendet die dynamische Linkmöglichkeit von OS/2. Wir erhalten eine Handle auf das Modul DOSCALLS mit Hilfe der Funktion loadmodule, linken diese (getprocadr) und rufen sie auf (call). Weder UR/Forth noch OS2XLisp überprüfen die Parameter, die Sie der OS/2-Funktion übergeben. In beiden Fällen kann aber der Anwender selbst eine Parameterüberprüfung implementieren.

Ein weiterer wichtiger Aspekt dieses kleinen Beispielprogramms ist die Übergabe eines Far-Zeigers auf einen Puffer. In C wandelt der Funktions-Prototyp den Zeiger buf zu PCHAR (Far-Zeiger auf char). In Modula-2 liefert die Prozedur ADR die Adresse einer Variablen oder Prozedur. Die (Call-)Funktion in OS2XLisp wandelt automatisch String-Zeiger in Far-Zeiger um. In Forth liefert CREATE (ähnlich BUF) automatisch die Adresse. Jedoch ist die Adresse der Offset von BUF innerhalb des Wörterbuchs von Forth. Um aber die ganze Adresse zu erhalten, benötigen wir noch den Segmentselektor des Forth-Datensegments, den das Forth-Wort DS0 liefert.

Ähnlich Forth behandelt auch Basic das Segment und den Offset unterschiedlich. Die Deklaration von DosGetModName% behandelt den Far-Zeiger wie zwei getrennte Parameter: VARSEG erhält das Segment und SADD liefert den Offset des Strings mit variabler Länge. Das Prozentzeichen am Ende des Funktionsnamens kennzeichnet, daß diese Funktion einen Integerwert zurückgibt.

Die Funktion DosGetModName liefert einen ASCIIZ-String in einem Puffer, aber ASCIIZ ist nicht der Stringtyp jeder Programmiersprache. In UR/Forth wandelt der Ausdruck BUF - ASCIIZ COUNT TYPE den Inhalt von BUF von ASCIIZ in einen Forth-String und gibt ihn dann aus. In Basic können wir den Befehl PRINT verwenden, müssen aber vorher den String mit RTRIM behandeln und den Rest des Puffers mit Leerzeichen füllen, bevor wir ihn verwenden. In Modula-2 sind die Strings ähnlich denjenigen von Pascal. Jeder String hat eine definierte Länge. Modula-2 versteht aber auch den 0-Delimiter. OS2XLisp ist in C geschrieben, und verwendet das C-ASCIIZ-Stringformat.

```
\ threads.4th
\ Andrew Schulman 23-July-1988
\ revised 27-July: using GETNUM from input stream, not stack
\ revised 3-August: using new TASKER.BIN (Ø8-Ø2-88)

TASKER

\ ==========================================================
\ note that each thread has its own view of the LDT
VARIABLE GIS   VARIABLE LIS
DSØ GIS DSØ LIS DOSGETINFOSEG DROP

: SELF ( -- thread# of current thread )   LIS @ 6 @L ;

: CHAR ( n -- char )   DUP 9 > IF 55 ELSE 48 THEN + ;

\ ==========================================================
\ stolen from os2demo.scr
VARIABLE SEED
: (RAND)   SEED @ 259 * 3 + 32767 AND DUP SEED ! ;
: RANDOM   (RAND) 32767 */ ;

: RY ( -- y ) ?YMAX 1+ RANDOM ;
: RX ( -- x ) SELF 4 MOD 2Ø * 2Ø + RANDOM SELF 4 MOD 2Ø * MAX;

\ ==========================================================
\ use UR/F semaphores, instead of directly calling OS/2
SEMAPHORE SEM

VARIABLE ID
VARIABLE ATTR   12 ATTR !

\ display thread ID -- multiple threads will execute through
\ this same code if the semaphore is removed, you can see
\ each thread grab wrong thread id#
: SHOW-ID ( -- )
   BEGIN
      SEM SGET
      SELF CHAR ID !
      \ use VioWrtCharStrAtt because it is properly
      \ reentrant and also does not screw with the cursor
      \ position
      DSØ ID  1  RY  RX  DSØ ATTR  Ø VIOWRTCHARSTRATT  DROP
      SEM SREL
      PAUSE
   AGAIN ;

\ ==========================================================
\ simulate the cobegin-coend construct, so all threads
\ start together
: COBEGIN ( -- )   DOSENTERCRITSEC DROP ;
: COEND ( -- )     DOSEXITCRITSEC DROP ;

\ ==========================================================
2Ø48 128 TCB THREAD1
2Ø48 128 TCB THREAD2
2Ø48 128 TCB THREAD3
2Ø48 128 TCB THREAD4

: (START) ( tcb -- )   DUP START SHOW-ID WAKE ;

: START-IT ( -- )
   COBEGIN
      THREAD1 (START)
      THREAD2 (START)
      THREAD3 (START)
      THREAD4 (START)
   COEND ;

\ ==========================================================
\ important to not stop thread while it's holding the semaphore
: (STOP) ( tcb -- )
   SEM SGET
   STOP
   SEM SREL ;
```

Listing 2: *Der Quellcode von Threads.4th.*

```
: STOP-IT ( -- )
   THREAD1 (STOP)
   THREAD2 (STOP)
   THREAD3 (STOP)
   THREAD4 (STOP) ;

\ ============================================================
\ get a number from input stream, put on stack
: GETNUM ( -- n )   BL WORD NUMBER? 2DROP ;

\ ============================================================
\ UR/F tasks are OS/2 threads, so we can manipulate a task
\ using OS/2 calls

\ important to claim semaphore, so we don't suspend a thread
\ while it's holding on to the semaphore !
: SUSPEND ( -- ) GETNUM SEM SGET DOSSUSPENDTHREAD SEM SREL DROP
;
: RESUME ( -- )  GETNUM DOSRESUMETHREAD DROP ;

\ if entered from keyboard, suspend/resume all background
\ tasks
: CRIT ( -- )       DOSENTERCRITSEC DROP ;
: XCRIT ( -- )      DOSEXITCRITSEC DROP ;

\ if entered from keyboard, also suspend/resume all
\ background tasks
: GETSEM ( -- )     SEM SGET ;
: RELSEM ( -- )     SEM SREL ;

\ ============================================================
VARIABLE PRTY

: GET-PRTY ( -- )   2 DSØ PRTY GETNUM DOSGETPRTY DROP PRTY @ .
;

: SET-IDLE ( -- )  2 1 Ø GETNUM DOSSETPRTY DROP ;
: SET-REG ( -- )   2 2 Ø GETNUM DOSSETPRTY DROP ;

\ warning:  if you set a background thread to time-critical
\ from the keyboard, you never get the keyboard back!

\ ============================================================
: HELP ( -- )
   CR ." Commands:" CR
   ." HELP" CR
   ." CRIT" CR
   ." XCRIT" CR
   ." GETSEM" CR
   ." RELSEM" CR
   ." SUSPEND <thread #>" CR
   ." RESUME <thread #>" CR
   ." GET-PRTY <thread #>" CR
   ." SET-IDLE <thread #>" CR
   ." SET-REG <thread #>" CR
   ." STOP-IT" CR ;

\ ============================================================
CLS
START-IT
THREAD1 (STOP)          \ leave room for typing in commands
PAUSE
CLS
HELP
```

Listing 2: *(Ende)*

UR/Forth

UR/Forth ist ein Multitasking-Threaded-Code-Interpreter. Im Gegensatz zu DOS-Aufrufen, die auf Registerwerten und Interrupts basieren, werden bei OS/2 Parameter auf dem Stack abgelegt, und ein Far-Call ruft die Funktion auf. Der Aufruf ist stackorientiert, paßt also gut zu Forth.

Beim Aufruf einer OS/2-Funktion von UR/Forth werden die Parameter auf dem Stack abgelegt, gefolgt vom Namen der OS/2-Funktion, die diese Parameter benötigt. Dies ist bei den anderen Versionen von Forth gleich. Sehen Sie sich folgende Anwendung an:

```
880 100 + . 980 ok
880 100 DOSBEEP . Ø ok
```

Ebenso wie die interaktive Eingabe von 880 100 +, um Forth als PRN-Rechner zu benutzen, können auch OS/2-Funktionen aufgerufen werden.

Da die OS/2-Funktionen ohne großes Programmieren aufgerufen werden können, ist UR/Forth ein nützliches Hilfsmittel, um mit OS/2 zu experimentieren und es verstehen zu lernen. So wie + die obersten beiden Parameter des Stacks durch die Summe der beiden ersetzt, ersetzt der Operator DOSBEEP die obersten beiden Parameter auf dem Stack mit dem OS/2-Fehlercode. In diesem Fall tritt zusätzlich der Nebeneffekt auf, daß es sich um die erste Note der Toccata von Bach handelt. Der Punkt ist nicht das Bereitzeichen, sondern das Forth-»Wort« für die Anzeige des obersten Werts auf dem Stack. Nachdem Forth ein Kommando ausgeführt hat, gibt es »ok« aus und wartet auf das nächste Kommando.

Was verbirgt sich hinter Forth? Es ist ein interaktiver Interpreter, Compiler und Assembler in einem. Forth verwaltet ein Wörterbuch, worin »+«, der Punkt und »DOSBEEP« enthalten sind. Jedes neue Wort, das Sie erzeugen, wie auch ENUMDLL in Abbildung 1, besteht aus vorher definierten Wörtern, die »zusammengefädelt« werden, oder aus Maschinenbefehlen, die der Forth-Assembler erzeugt.

Ein äußerer Interpreter liest Eingaben von der Tastatur oder aus einer Datei, ein Compiler verkettet die Anweisungen und ein innerer Interpreter rast die Kette der Anweisungen entlang und führt sie aus. Der Anwender hat zu jeder dieser Komponenten direkten Zugang (z.B. die Verwendung des äußeren Interpreters in einem eigenen Programm).

UR/Forth unter OS/2 ist eine geschützte OS/2-Task, kein eigenes Betriebssystem, wie einige andere Forth-Implementierungen. UR/Forth ist für Multitasking- und geschützte Speicher-Umgebungen entwickelt. Zusätzlich zu dem direkten Zugriff auf OS/2 über das API bietet UR/Forth ein maschinenunabhängiges mehrschichtiges Systeminterface, so daß Sie auch MALLOC statt DOSALLOCSEG oder FOPEN statt DOSOPEN verwenden können.

Erstellen von Forth-Wörtern

Die Beispiele in Abbildung 1 sind sich so ähnlich wie möglich. In Forth kann jemand sogar dieses Miniprogramm in mehrere Worte unterteilen. Alles, außer einer Zahl, ist in Forth ein Wort. Dies gilt auch für Kontrollstrukturen. Sie können also neue Kontrollstrukturen und eine benutzerprogrammierbare Syntax programmieren. Forth-Worte werden mit dem Wort : (einem Doppelpunkt) definiert. Ein ;

(Semikolon) beendet die Definition. In Abbildung 1 definieren wir ENUMDLL, welches ein DO LOOP enthält. Die Ausführung wird durch die Eingabe von ENUMDLL gestartet.

Wir können ENUMDLL in andere Worte integrieren, aber es erledigt zu viel, um universal verwendbar zu sein. Eine realistischere Version von Abbildung 1 sehen Sie in *Listing 1*. Eines unserer neuen Worte MODULE? erwartet eine Zahl auf dem Stack, testet diese mit DOSGETMODNAME und hinterläßt ein boolesches Flag auf dem Stack. Dieser (vorher - nachher) Stackzustand wird im geklammerten Kommentar festgehalten, welches eine Forth-Konvention für das Kommentieren von Worten ist.

Das neue Wort .ASCIIZ gibt einen ASCIIZ-String aus, dessen Adresse auf dem Stack liegt. Da das Forth-Wort . (Punkt) etwas ausgibt, ist oftmals das erste Zeichen eines Wortes, das eine Ausgabe vornimmt, der Punkt. Das Wort .MODULE gibt den Modulnamen und die Nummer aus. Es ruft unser eben definiertes Wort .ASCIIZ auf. Die neue Version von ENUMDLL verwendet die Worte MODULE? und .MODULE.

Die Struktur von IF in Forth ist etwas gewöhnungsbedürftig, aber konsistent mit der Postfix-Notation von Forth. Beachten Sie, daß THEN hinterher bedeutet, und keine Verbindung zu dem Sprachkonstrukt if..then anderer Sprachen besitzt.

Die Zusammenarbeit mit Threads

Es gibt nichts bemerkenswertes zu diesem Programm. Wie Sie in Abbildung 1 sehen, kann dieses Programm auch in C, Lisp, Modula-2 und Basic geschrieben werden. *Listing 2* zeigt eine interessantere Seite von Forth unter OS/2, ein kleines interaktives Labor zum Experimentieren mit mehreren Threads.

Wenn Sie mit Forth vertraut sind, wird es Sie überraschen, das Listing 2 in UR/Forth durch den einfachen Befehl INCLUDE THREADS.4TH laufen kann. Zusätzlich zum alten Stil mit den 1024 Byte großen Screenblöcken, die in den meisten Forth-Implementierungen verwendet werden (für die UR/Forth einen guten Editor bietet) versteht UR/Forth auch die INCLUDE-Anweisung, um Forth-Sourcecode aus normalen Textdateien zu übersetzen.

In der Datei THREADS.4TH werden mehrere Threads aktiviert, wobei jeder denselben Programmteil ausführt und seine Thread-ID auf einem bestimmten Teil des Bildschirms ausgibt. Ein Teil des Bildschirms bleibt für Ihre Kommandoeingabe frei und eine Hilfemeldung wird angezeigt. Werden die Threads 5, 6 und 7 angezeigt, so können Sie zum Beispiel SUSPEND 6 eingeben und Thread 6 wird angehalten. Nach der Eingabe von RESUME 6 startet er wieder. Nach der Eingabe von SET-IDLE 6 scheint er wieder anzuhalten. Aber nach der Eingabe von SUSPEND 5 und SUSPEND 7 beginnt Thread 6 wieder seine Bildschirmausgabe, da er mehrere Zeitscheiben zugeteilt bekommt.

Sie sehen, daß es keinen Funktionsaufruf von DOSCREATETHREAD gibt. Es sind aber Threads vorhanden. Wie viele andere Versionen von Forth, bietet auch UR/Forth Multitasking-Fähigkeiten, die das Starten von Hintergrundprozessen unterstützen. In den Versionen für MS-DOS und den 386er werden die Forth Task-Kontrollblöcke und die Semaphore komplett von Forth verwaltet.

In der neuesten Version von UR/Forth hat LMI aber die Tasks von Forth als Threads unter OS/2 und die Forth-Semaphore als OS/2-Semaphore implementiert (UR/Forth wird mit großen Teilen im Quellcode ausgeliefert, somit können Sie auch sehen, wie der Multitasker arbeitet). Die Forth-Tasks unter OS/2 sind also vollständig zeitscheibengesteuert, wobei in anderen Implementierungen die Umschaltung davon abhängt, ob jemand PAUSE aufruft. Dies erinnert an die nicht zeitscheibengesteuerten Umgebungen Microsoft Windows oder Apple MultiFinder. Neben dem Einschluß der API-Schnittstelle ist dies der größte Unterschied der OS/2-Version und der anderen Versionen von UR/Forth.

Wir können die Standard-Multitasking-Worte von UR/Forth (wie TCB zum Erzeugen eines Task Control Blockes, SLEEP zum Anhalten einer Task oder SGET zum Belegen eines Semaphors) benutzen, und wissen, daß wir wirklich die OS/2-Multitasking-Funktionen von OS/2 verwenden. Andererseits können wir die OS/2-Funktionsaufrufe wie DOSSUSPENDTHREAD benutzen, um die Forth Task-Kontrollblöcke zu manipulieren.

Jede Task sollte irgendwo in seiner Hauptschleife PAUSE aufrufen, da Pause zum Löschen eines Threads dient, wenn eine Task durch STOP beendet wird. Da OS/2 keinen DosKillThread-Aufruf kennt, muß STOP eigenartige Dinge unternehmen, um einen Thread zu überzeugen, daß er DOSEXIT aufrufen soll.

Obwohl UR/Forth ein Interpreter ist, enthält es einen Compiler und einen Assembler. Es kann daher Objektcode erzeugen, dessen Adresse an DOSCREATETHREAD übergeben werden kann. Bedenken Sie, daß alle Threads denselben Code ausführen. Jeder Thread erhält seine eigene ID, da jeder Thread sein eigenes lokales Informationssegment erhält.

THREADS.4TH erzeugt eine eigene kleine Kommandosprache zum Spielen mit Threads/Tasks. In Forth ist es eine triviale Angelegenheit, solche kleinen »Sprachen« zu erzeugen. Anstatt den Benutzer zu zwingen, sich der Postfix-Notation von Forth anzupassen und Kommandos wie 6 SUSPEND einzugeben, definieren wir in diesem Beispiel GETNUM, welches eine Zahl vom Eingabestrom und nicht vom Stack holt. Aus diesem Grund können Sie SUSPEND 6 eingeben. Dies zeigt auch, wie einfach es für einen Forth-Programmierer ist, den Sprachparser in seinen eigenen Applikationen zu verwenden - was Sie mit einem C-Compiler nicht können.

```
\ xy.4th for UR/Forth
\ get current cursor location, by programming the 6845 CRTC

8Ø CONSTANT COLUMNS

HEX
: CUR-LOC (-- reg14 reg15) E 3D4 PC! 3D5 PC@ F 3D4 PC! 3D5 PC@;
: MK-WORD (w1 w2 -- w) SWAP FF * + ;
: COL-ROW (w -- x y )  DUP COLUMNS MOD 1+  SWAP  COLUMNS / 1+ ;
: CURS  CUR-LOC  MK-WORD  COL-ROW ;
DECIMAL
```

Listing 3: *Der Quellcode von xy.4th.*

Grafik, IOPL und Ports

UR/Forth für OS/2 bietet einige gute Grafikfunktionen. Sie können in dieser Umgebung gute, nicht auf dem Presentation Manager basierende Grafikapplikationen schreiben. Die Grafik kann innerhalb der UR/Forth-Tasks benutzt werden, womit Sie in der Lage sind, Multitasking-Grafikprogramme zu schreiben.

Ein Grund, warum die Grafikprogrammierung unter OS/2 schwierig ist, ist der IOPL-Schutz, der den Zugriff auf die Ein-/Ausgabeports schwierig gestaltet. UR/Forth erleichtert dies sehr, wie das kleine Programm XY.4TH in *Listing 3* zeigt. XY.4TH programmiert den Videokontroller 6845 direkt, um die aktuelle X- und Y-Position des Cursors auf dem Stack abzulegen. UR/Forth enthält auch die Funktion ?XY, die dasselbe durch einen Aufruf von VIOGETCURPOS erledigt. XY.4TH hilft aber einige Punkte zu verdeutlichen.

Zum Senden eines Bytes an einen Port des PCs können Sie in UR/Forth PC! verwenden. Das Lesen eines Werts von einem Port erledigt die Funktion PC@. In unserem Beispiel schicken wir 0EH an den Port 3D4H und lesen dann den Wert vom Port 3D5H. Beim ersten Gebrauch einer dieser beiden Funktionen leuchtet die Disklampe, da OS/2 das IOPL-Segment laden muß. Jeder Aufruf von PC@ und PC! erzeugt einen Far-Call in das IOPL-Segment, deshalb sind sie nicht so schnell, wie die Versionen in Real-Modus-UR/Forth. Sie brauchen DOSPORTACCESS nicht aufzurufen, da es für alle Ports bei der Initialisierung von UR/Forth aufgerufen wird.

Das Wort CUR-LOC legt die Inhalte der Register 14 und 15 des 6845-Kontrollers auf dem Stack ab. MK-WORD wandelt das Byte in ein Wort, und COL-ROW errechnet die augenblickliche X- und Y-Position des Cursors. Diese drei Worte werden in der Definition von CURS »gefädelt«. Alles geschieht ohne Variablen. Die gesamte Kommunikation geschieht auf dem Stack. Aus diesem Grund können Forth-Worte ohne weiteres mehrere Ergebnisse zurückliefern - wieder etwas, das in C nur simuliert werden kann.

Eine weitere Kostbarkeit von UR/Forth ist die Datei OS2.SCR, die eine Demonstration von gemeinsamem Speicher, Queues, Pipes und so weiter ist. Eine Datei mit dem Namen URPOUN.ARC gestattet es Ihnen, OS/2-Datenstrukturen zu erzeugen. LMI BBS vertreibt auch noch Applikationen wie SSE (einen Forth Decompiler), ein Turtlegrafik-Paket und URTCO, einen Optimizer für Threaded-Code.

Was bedeutet eigentlich UR? Fragen Sie mich nicht!

OS2XLisp

In einigen Dingen ist Lisp das Gegenteil von Forth. In Forth gaben wir zum Beispiel 880 100 + . ein, wogegen wir im OS2XLisp-Interpreter den folgenden Ausdruck (das Lisp Bereitzeichen ist die spitze Klammer) eingeben müssen:

```
> (+ 88Ø 1ØØ)
98Ø
```

Die meisten Programmiersprachen verwenden die Infix-Notation (880 + 100), Forth verwendet die Postfix-Notation, die Lisp-Funktionen hingegen stehen am Beginn der Liste.

Ein Lisp-Funktionsaufruf besteht aus einer Liste, die durch die berüchtigten Lisp-Klammern zusammengefaßt wird. Der Lisp-Bewerter benutzt das erste Element als Funktion (car) die auf die restlichen Elemente der Liste (cadr) angewendet wird. In obigem Beispiel wird + auf 880 und 100 angewendet. In Forth mußten wir explizit die Ausgabe auf dem Bildschirm veranlassen. Der Lisp-Interpreter erledigt dies hingegen automatisch. Die oberste Ebene von Lisp ist eine Lese-/Auswerte-/Ausgabeschleife.

Forth und Lisp sind aber nicht gänzlich verschieden. Beide benutzen die Methode, daß in einem Ausdruck die Funktion oder der Operator an einem bestimmten Platz stehen muß. Auch ist OS2XLisp wie UR/Forth interaktiv. Sie können also OS/2-Systemaufrufe eingeben und unmittelbar das Ergebnis sehen.

Die Klammerung von Lisp erlaubt es, einfach Listen zu verschachteln, wodurch das Ergebnis einer Funktion zum Parameter einer anderen wird. Im folgenden Beispiel übergeben wir das Ergebnis einer Addition an ein Makro, das Variablen erzeugt:

```
> (define tmp (+ 88Ø 1ØØ))
98Ø
```

Zur Anzeige eines Objekts in Lisp brauchen Sie nur seinen Namen ohne Klammern einzugeben:

```
> tmp
98Ø
```

XLisp erziehen

Versuchen wir dieselben Dinge wie in Forth. Wir wollen sehen, ob DOSBEEP genauso funktioniert, wie + (dies ist unser Test, ob ein leichter Aufruf des API möglich ist):

```
> (dosbeep 88Ø 1ØØ)
error: unbound function - DOSBEEP
```

Oh je! OS2XLisp weiß nicht einmal, was DosBeep ist!

Tatsächlich weiß OS2XLisp wenig über das API von OS/2. Wir können aber XLisp beibringen, was DosBeep ist.

OS2XLisp kennt die DLLs, und dies ist der Hauptunterschied zu XLisp, welches unter anderen Betriebssystemen läuft. Begnügen wir uns auf die Schnelle, indem wir versuchen, OS2XLisp dosbeep beizubringen, während es läuft:

```
> (define doscalls loadmodule "DOSCALLS"))
1910
> (define dosbeep
      (getprocaddr doscalls "DOSBEEP"))
121654475
```

Was geht hier vor? Wir haben eine numerische Variable dosbeep erzeugt und sie mit einer großen Zahl initialisiert. Was bedeutet diese Zahl aber? Vielleicht sieht sie in Hexadezimaler Darstellung vernünftiger aus:

```
> (ultoa dosbeep 16)
"48830000"
```

C-Programmierer werden die Funktion ultoa aus der Standardbibliothek kennen. Bei der Anzeige sehen Sie, daß es sich um eine Speicheradresse handelt - die Prozeduradresse von DosBeep, um genau zu sein. Die numerische Lisp-Variable dosbeep ist ein Zeiger auf eine Funktion, über den wir die Funktion aufrufen können:

```
> (call dosbeep(word 880) (word 100))
; Lautsprecher ertönt
```

Es funktioniert, aber eigentlich wollten wir (dosbeep 880 100) analog zu (+ 880 100) eingeben. Indem wir die 4 Byte langen Lisp-Zahlen in 2 Byte-Werte konvertieren müssen, bereitet uns DosBeep zu viele Probleme, und somit müssen wir es explizit aufrufen (call).

Wir können eine neue Lisp-Funktion programmieren, die uns das Gewünschte erledigt:

```
> (define(dosbeep freq dur)
    (call dosbeep (word freq)
    (word dur)))
    DOSBEEP
> (dosbeep 880 100)
; Lautsprecher ertönt
0
```

Da Sie glauben werden, daß der Zugriff auf OS/2 wie ein Patch zur Laufzeit ist, werde ich Ihnen jetzt erklären, wie OS/2 in Lisp integriert ist. Stellen Sie sich vor, daß eine Note eine Liste von zwei Werten ist: Frequenz und Dauer. Wir können dem Symbol »bach« eine Liste von Noten zuordnen und eine Funktion (play) programmieren, die für jede Note einer Liste die Funktion dosbeep aufruft:

```
(define bach '((880 100)
    (784 100) (880 1200)))
(define (play notes)
    (mapcar
        (lambda (note)
            (DOSBEEP
            (car note)
            (cadr note)))
> (play bach)
; da-da-dum
(0 0 0)
```

In der Funktion (play) bildet (mapcar) die Liste der Noten ab und ruft für jede Note in der Liste eine Funktion auf. Diese Funktion hat keinen Namen und ist temporär erzeugt durch (lambda). Diese nimmt eine Note und übergibt sein erstes Element (car) an (dosbeep) als Frequenz, und sein zweites Element (cadr) als Dauer. Dies entspricht einem C-Programm, das strcmp an qsort übergibt.

Variablen und Symbole

Wir haben (define) sowohl zum Erzeugen von Variablen als auch von Funktionen benutzt. Dies ist vom Scheme-Dialekt von Lisp ausgeborgt. Wenn Sie aber mit Common Lisp vertraut sind oder nach einem Common Lisp-Buch arbeiten, können Sie auch (setf) und (setq) für die Variablen und (defun) für Funktionen benutzen.

Es ist wichtig zu wissen, daß es keinen großen Unterschied zwischen Funktionen und Daten gibt. Dosbeep ist eine Zahl, aber ebenso eine Funktion. In Lisp arbeiten Sie mit Symbolen, nicht mit Variablen. Symbole können verschiedene Werte beherbergen. Das Symbol dosbeep zum Beispiel besitzt einen numerischen Wert (die Adresse von DosBeep) und einen Funktionswert (den Programmteil, den wir gerade definiert haben). Ein Symbol kann auch eine Eigentumsliste besitzen (Microsoft Windows hat die Idee der Windows-Eigentumsliste von Lisp übernommen).

Man kann sich die Implementierung dieser Objekte unschwer vorstellen, wenn man all die Möglichkeiten der Datenobjekte kennt. In der C-Implementierung von XLisp stellt ein Lisp-Datenobjekt eine C-Union dar, die durch den Objekttyp unterschieden wird und Werte für Zahlen, Funktionswerte, Objektnamen, Eigentumsliste und so weiter beherbergen kann. Da diese Symbole verkettet werden können, besitzt das Symbol auch einen Zeiger zum nächsten Objekt.

In OS2XLisp wird sogar eine einfache Zahl wie 880 als solch komplizierte Datenstruktur verwaltet. Wenn wir 880 an die Funktion (dosbeep) übergeben, übergeben wir tatsächlich einen Zeiger auf diesen XLISP-Knoten.

Dynamisches Laufzeitlinken

Da die Lisp-Objekte so verschieden sind von »normalen« Variablen, wäre OS/2 ganz schön verwirrt, wenn es dies übergeben bekäme. So müssen Konvertierungen von Lisp nach OS/2 und zurück durchgeführt werden.

Als ich begann, XLisp nach OS/2 zu portieren, wollte ich für jeden OS/2-Funktionsaufruf eine Lisp-Funktion schaffen. Mir wurde aber schnell klar, daß ich durch die benötigten Konvertierungen für jede Funktion eine riesiges OS2XLisp.EXE erhalten hätte. Da ich auch faul bin, wollte ich nicht einige Hundert Funktionen eintippen.

OS/2 unterstützte mich durch das dynamische Laufzeitlinken. Dies ist ziemlich unterschiedlich vom in OS/2 verwendeten dynamischen Linken. Mit den OS/2-Funktionen DosLoadModule und DosGetProcAddr (Windows-Programmierer kennen diese Funktionen unter dem Namen

LoadLibrary und GetProcAddress), kann ein Programm ASCIIZ-Strings an OS/2 übergeben und einen Funktionszeiger zurückerhalten. Die Funktion kann dann über diesen Zeiger aufgerufen werden. In einigen Programmiersprachen wird dies »Stringaufruf« genannt.

Durch das dynamische Laufzeitlinken kann OS2XLisp durch einige Funktionen alle OS/2 API-Funktionen aufrufen. Die Funktion (loadmodule) liefert eine Modulhandle einer dynamischen Linkbibliothek (DLL). (getproc) liefert einen Zeiger auf eine Funktion in der DLL. (call) ruft eine Funktion über einen Zeiger auf. (c-call) erledigt dasselbe für alle Funktionen, die die CDECL-Konventionen für den Aufruf benötigen. (freemodule) beendet den Zugriff auf die DLL.

Sehen wir uns ein Beispiel an:

```
> (define dosgetinfoseg
    (getprocaddr doscalls
        "DOSGETINFOSEG"))
1201340416
```

Die Funktion DosGetInfoSeg erwartet, daß sie mit zwei Parametern aufgerufen wird. Der erste ist ein Zeiger auf 2 Bytes (ein Wort), in die OS/2 den Selektor des globalen Infosegments (GIS) schreibt. Der zweite ist ein Zeiger auf ein Wort, in das OS/2 den Selektor des lokalen Infosegments (LIS) schreibt. Wir müssen auch einige 2 Byte große Variablen für OS/2 erzeugen:

```
> (setf gis (word 0) lis (word 0))
0
```

Ich verwende (setf) hier anstatt von (define), da es auf einmal mehrere Zuweisungen ausführen kann. XLisp-Zahlen sind immer 4 Byte groß, weswegen wir die OS2XLisp-Funktion (word) zum Konvertieren benützen. Um die Funktion aufzurufen, dürfen wir nicht gis und lis, sondern deren Adressen übergeben. Wir verwenden das Makro ^, das eine Abkürzung für die OS2XLisp-Funktion (addr) darstellt. Dieses liefert einen Zeiger auf das Datum des XLisp-Knotens:

```
> (call dosgetinfoseg ^gis ^lis)
0
```

Die (call)-Funktion liefert 0 (ok), da dies DosGetInfoSeg liefert. Wie könnte DosGetInfoSeg auch nicht funktionieren? Jetzt können wir uns die Inhalte unserer Lisp-Variablen ansehen:

```
> gis
96
> lis
15
```

OS/2 hat diese Lisp-Objekte einfach geändert.

Diese Werte für sich sind in der Regel wertlos. Da aber OS2XLisp den geschützten segmentierten Speicher kennt, können wir nicht nur den Inhalt dieser Segmente, sondern auch die Segmentattribute ansehen. (lsl) entspricht zum Beispiel dem Befehl LSL im geschützten Modus, der den letzen gültigen Offset des Segments liefert (die Größe minus 1):

```
; chat.lsp
; Run-time dynamic linking to CHATLIB.DLL from OS2XLISP

(define chatlib (loadmodule "CHATLIB"))
(define login (getprocaddr chatlib "LOGIN"))
(define get-msg-cnt (getprocaddr chatlib "GET_MSG_CNT"))
(define my-id (call login))
(define msg-cnt (call get-msg-cnt (word my-id)))
```

Listing 4: *Dynamisches Linken mit einer DLL aus OS2XLisp.*

```
> (lsl gis)       ; wie groß ist es
69                ; Größe - 70 Bytes
> (verr gis)      ; Ansehen
T                 ; ja
> (verw gis)      ; Habe ich Zugriff?
NIL               ; Nein
```

NIL ist in Lisp das Symbol für die leere Liste oder einfach FALSE. T ist das Symbol für TRUE. Die Funktionen (verr) und (verw) entsprechen den Befehlen im geschützten Modus, die wir später noch besprechen.

Wie mit UR/Forth müssen wir zum Ansehen dieser Infosegmente einen Far-Zeiger bilden. Stellen Sie sich vor, wir wollen die Prozeß-ID des Prozesses, der im Vordergrund läuft. Da wir mit OS2XLisp arbeiten, ist dies der Vordergrundprozeß. Die Information finden wir an den 2 Byte ab Offset 28 von GIS:

```
> (peek (mk-fp gis 28) 'word)
11
```

(mk-fp) nimmt als Argumente ein Segment und einen Offset und erzeugt daraus einen Far-Zeiger. Die Funktion (peek) benötigt einen Far-Zeiger und eine optionale Direktive, die angibt, was zu holen ist. Sie können Bytes, Worte, Long-Zahlen, IEEE-Fließkommazahlen und ASCIIZ-Strings laden. Das Anführungszeichen vor einem Lisp-Symbol gibt Lisp zu erkennen, daß wir wirklich am Symbol selbst interessiert sind, nicht an einem möglichen Inhalt.

Kommunikation mit DLLs

Das dynamische Laufzeitlinken stellt sich als großes Plus heraus. Es ist nicht nur möglich alle OS/2-Funktionen durch einige Funktionen aufrufen zu können, anstatt durch einige Hundert, sondern es bedeutet auch, daß der Anwender alles aus dynamischen Linkbibliotheken aufrufen kann, nicht nur die APIs von DOSCALLS, VIOCALLS und KBDCALLS. Beim Erscheinen neuer Funktionen sind diese sofort in OS2XLisp zugänglich, ohne auf einen Update warten zu müssen.

Listing 4 zeigt den Aufruf von CHATLIB.DLL durch OS2XLisp. OS2XLisp kannte CHATLIB.DLL vorher nicht. Alle Verbindungen wurden zur Laufzeit geknüpft, was ein einfacher Weg zum Erstellen eines erweiterbaren Interpreters ist. Die Aufgabe des Linkers verschieben wir bis zum letzten Augenblick. Alle guten Programmierer wissen:


Above™ Board Plus
intel
Word
Microsoft Excel
Vollständiges Tabellenkalkulationsprogramm mit Grafik und Datenbank
Für IBM Personal System/2, IBM PC/AT und Kompatible
Enthält 5¼-Zoll (1,2 MByte) und 3½-Zoll Disketten

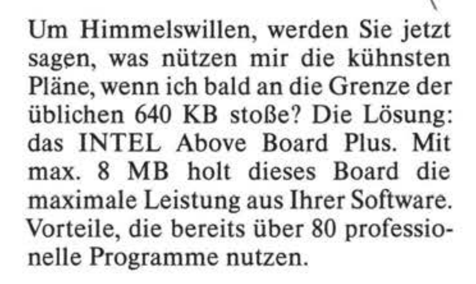
Ein Geschenk des Himmels!

ZWEITER TEIL

Jetzt öffnen sich wieder die Himmelspforten. Ab sofort gibt es das neue Intel Above Board Plus zweimal zum himmlisch günstigen Preis. Das erste heißt Management-Package, besteht aus Above Board Plus, Microsoft Excel und einer Microsoft-Maus.
Das zweite ist das Windows/286-Package und enthält das Board inclusive dem Microsoft Windows/286.
Beide Kombinationen bringen Ihnen den Himmel auf Erden: Zeit- und Geldersparnis durch zwei Pakete, die teuflisch gut zusammen passen. Von zwei führenden Unternehmen der Branche.

1. Das Management-Package

Für die Entscheider von morgen heißt es nicht mehr nur, Fakten und Zahlen zu sammeln. Es gilt, Trends aufzuspüren und Analysen zu erstellen. Dafür brauchen Entscheider das beste Werkzeug. Sie brauchen das Management-Package. Excel nutzt erstmals alle Vorteile von Windows, erleichtert so den späteren Übergang zu OS/2 und übernimmt mühelos alle Daten aus Lotus, dBase, Multiplan etc. Excel vereint Kalkulation, Grafik und Datenbank: So werden aus Zahlen Strategien und aus Informationen Entscheidungen. Und weil die neue Microsoft-Maus serienmäßig zum Package dazugehört, wird nicht nur die Bedienung noch einfacher, sondern das Paket auch himmlisch günstig:
Sie sparen 435,- DM.



Above Board Plus.
Mehr Raum für kühne Pläne.

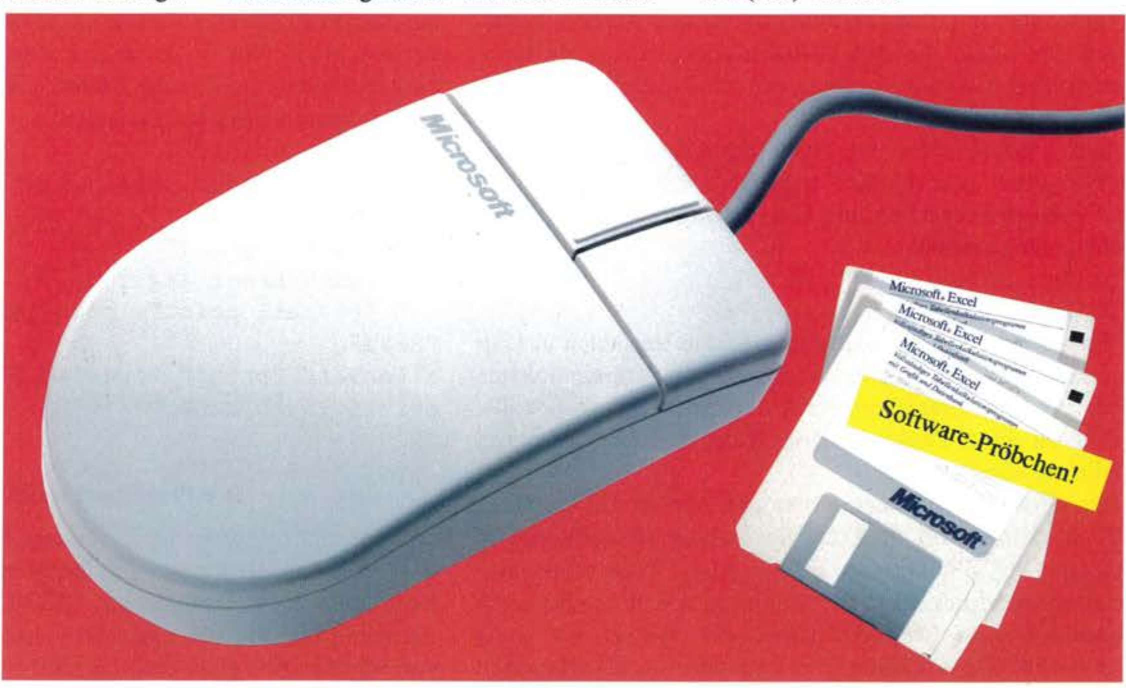
Um Himmelswillen, werden Sie jetzt sagen, was nützen mir die kühnsten Pläne, wenn ich bald an die Grenze der üblichen 640 KB stoße? Die Lösung: das INTEL Above Board Plus. Mit max. 8 MB holt dieses Board die maximale Leistung aus Ihrer Software. Vorteile, die bereits über 80 professionelle Programme nutzen.

2. Das Windows/286-Package

Sie brauchen mehr Speicher, setzen auf INTEL-Qualität und hätten gern Multitasking unter WINDOWS/286? Unser zweites himmlisches Erlebnis, die neuen Above Board Plus Hardware-Register für EMS 4.0 machen all dies möglich: Ab sofort gibt es zu jedem Above Board Plus das neue Windows/286. Und 5 Jahre Garantie.
Sie sparen 450,-DM.

Schauen Sie also schnell bei Ihrem Händler vorbei. Die Angebote gelten, solange der Vorrat reicht. Und es wäre doch die Hölle, würden Sie zu spät kommen. Wir nennen Ihnen gerne einen Fachhändler in Ihrer Nähe.

COMPUTER 2000 AG
Baierbrunner Str. 31
8000 München 70
Tel. (089) 76990-0





Microsoft



Wir wissen, was läuft.

»Verschiebe ruhig auf Morgen, was du heute könntest besorgen«.

Wir verwendeten OS2XLisp interaktiv, indem wir Ausdrücke direkt über die Tastatur eingegeben haben. Sie können aber auch Textdateien durch die Funktion (load) auswerten. Wäre zum Beispiel der Programmteil von Listing 4 in einer Datei mit dem Namen CHATLIB.LSP, können Sie Lisp mitteilen:

```
> (load 'chatlib)
```

Der Aufruf von C aus XLisp

OS2XLisp kann auch C-Funktionen in der dynamischen Linkbibliothek CRTLIB.DLL, die bei OS2XLisp enthalten ist, aufrufen. Früher benutzten wir ultoa, das in Lisp (ultoa) heißt:

```
> (define crtlib (loadmodule "CTRLIB"))
2520
> (define ultoa
    (getprocaddr crtlib "_ultoa"))
97984426
> (define (ultoa val base &aux (str
    (make_string 32 80)))
    (string-upcase
    (c-call ultoa val str
    (word base) 'str)))
ULTOA
> (ultoa dosgetinfoseg 16)
"479B0000"
```

Wie LINK, unterscheidet (getprocaddr) Groß-/Kleinschreibung. Funktionen, die den Pascal-Aufrufkonventionen folgen (wie etwa DosBeep oder KbdStringIn) sollten an (getprocaddr) in Großschreibung übergeben werden. Funktionen wie ultoa, die den C-Aufrufkonventionen entsprechen, sollten in Kleinschreibung mit vorangestelltem Unterstrich erscheinen.

Das Symbol ultoa hat einen numerischen und einen Funktionswert. Wir können sogar den numerischen Wert der Funktion übergeben, um die Adresse der Funktion im Speicher auszugeben:

```
> (ultoa ultoa 16)
"5D71FAA"
```

Es gibt mehrere Möglichkeiten, lokale Variablen in Lisp zu erzeugen. Ich verwende hier &aux; str ist ein lokaler String mit 80 Leerzeichen. Nach der Übergabe an (c-call), der ihn an die C-Funktion ultoa übergibt, enthält er etwas nützliches, nämlich "5D71FAA".

Da ultoa drei Argumente benötigt, warum wird dann (c-call) ein viertes Argument, das Symbol 'str übergeben? Die meisten OS/2-Funktionen geben einen Fehlercode unsigned short zurück. Bei einer Funktion wie ultoa, die einen Zeiger auf einen Character zurückgibt, müssen wir (call) oder (c-call) dies mitteilen. Die Funktion (c-call) gibt einen String von ultoa zurück. Dieser String wird der XLisp-Funktion (string-upcase) übergeben.

```
; makecall.lsp
; use Lisp symbolic manipulation
; to create OS/2 API functions
; "I would rather write code to write code, than write code"
; Andrew Schulman 2-August-1988

; filter sets up formal argument list to be passed to (define)
(define (filter list &aux (lst (list nil)) elem)
    (dotimes
        ; for each element in list
        (i (length list))
        (setf elem (nth i list))
        ; if it's a list, like (word freq)
        (if (listp elem)
            ; if it's not a retval directive, like 'word
            (if (not (eq 'QUOTE (car elem)))
                (nconc lst (list (cadr elem))))
        ;else
            (nconc lst (list elem))))
    (cdr lst))

(defmacro make-call (module func &rest args)
    '(define ,(append (list func) (filter args))
        (call
            ,(getprocaddr
                (eval module)
                (if (eq module 'CRTLIB)
              (strcat "_" (string-downcase (symbol-name func)))
                    (symbol-name func)))
            ,@args)))

; e.g.:
;       (make-call doscalls dosbeep (word freq) (word dur))
; produces:
;       (define (dosbeep freq dur)
;           (call <procaddr> (word freq) (word dur)))
; can then be called:
;       (dosbeep 880 100)
```

Listing 5: *Der Quellcode von makecall.lsp.*

Sogar die bekannteste C-Funktion printf, die eine variable Anzahl Parameter verwendet, kann von Lisp aufgerufen werden. Auch dies ist durch den Zauber des dynamischen Laufzeitlinkens (und einige kleinere Wunder der Common Lisp defmacro-Fähigkeit) möglich:

```
> (define printf
    (getprocaddr crtlib "_printf"))
97980568
> (defmacro printf
    (mask &rest args)
    '(c-call printf, mask ,@args))
PRINTF
> (printf "printf lives at %Fp\n" printf)
printf lives at 05D7:1098
28
```

Der Preis der Flexibilität

Obwohl uns (getprocaddr) sehr viel Flexibilität gab, ist es doch mühsam, sie jedesmal aufzurufen, wenn wir eine neue Funktion wollen. Deshalb hat OS2XLisp eine (enumproc) Funktion, die alle von einer DLL exportierten Funktionen aufzählt. Es gibt auch eine (install)-Funktion, die zu jeder von der DLL exportierten Funktion eine procaddr-Variable erzeugt:

```
> (install "CHATLIB" t)
SEND_MSG GET_MSG_CNT
LOGIN GET_MSG
LOGOUT T
```

Die install-Funktion arbeitet auch mit dem DOSCALLS-»Modul«, welches in Wirklichkeit der OS/2-Kern ist. Bei DOSCALLS.DLL-Funktionen arbeitet DosGetProcAddr nur mit gewöhnlichen Zahlen, aber OS2XLisp enthält eine Tabelle der Funktionsnamen von DOSCALL.

Da ein Lisp-Programm nur aus Lisp-Knoten besteht, die in einer Liste verkettet sind, ist es einfach, Programme zu schreiben, die Programme erzeugen. Wenn wir eine Lisp-Funktion schreiben wollen, die auf einer DLL-Funktion basiert, wenden wir immer dieselben Schritte an. Wir holen die Adresse der Funktion und rufen diese mit (call) oder (c-call) und den benötigten Parametern auf. Lisp kann dies aber mit dem kurzen Makro (make-call), das Sie in *Listing 5* sehen, für uns erledigen.

Die Funktion (make-call) nimmt eine OS/2 Modul-Handle, den Namen der Funktion und alle für diese Funktion benötigten Parameter. Sehen Sie ein Beispiel:

```
> (make-call viocalls viogetphysbuf buf
    (word handle))
    VIOGETPHYSBUF
```

OS2XLisp hat eine neue Funktion erzeugt (viogetphysbuf), die die zwei Parameter buf und handle benötigt, und die beim Aufruf die OS/2-Funktion VioGetPhysBuf aufruft. Bedenken Sie, daß der (getprocaddr)-Aufruf nur einmal erfolgt, wenn (viogetphysbuf) erzeugt wird, nicht bei jedem Aufruf von (viogetphysbuf).

Bis jetzt haben die DLL-benutzenden Funktionen, die wir verwendet haben, nur einfache Parameter benötigt. Wie verhält es sich mit OS/2-Datenstrukturen? Zum Beispiel benötigt (viogetphysbuf) einen Zeiger auf die Struktur PhysBufStruct. Die OS2XLisp-Funktion (make-struct), die in Lisp geschrieben ist, erzeugt Datenstrukturen von symbolischen Anweisungen in Lisp-Listen, und (unpack-struct) erzeugt Lisp-Listen aus OS/2-Datenstrukturen:

```
; Die Strukturbeschreibung
(define PhysBufStruct
    '((ptr bufaddr)
    (long size)
    (word sel)
    (word sel2)))
; Eine initialisierte Struktur
(define physbuf
    (make-struct PhysBufStruct
    '(#xb8000 #x4000 0 0)))
```

Jetzt können wir die Funktion aufrufen, die (make-call) für uns erzeugt hat:

```
> (viogetphysbuf physbuf 0)
0
> (define sel (assoc 'sel (unpack-struct
    PhysBufStruct physbuf)))
(SEL 847)
> (cadr sel)   ; 2. Element der Liste holen
847
```

Wir können überprüfen, daß 847 wirklich ein Selektor des physikalischen Screenbuffers ist:

```
> (define (writechar x y char)
    (poke (mk-fp 847 (* 2 (+ x (* 80 y))))
char))
WRITECHAR
> (writechar 79 24 #\a)
97
```

Dies schreibt ein a in die untere rechte Ecke eines CGA-Bildschirms.

Schutz-Verletzungen

Wie schützt sich OS2XLisp vor Schutz-Verletzungen? Da wir gerade (poke) benutzt haben, werden Sie wohl überlegt haben, wie sich der geschützte Modus bei Peek und Poke verhält. Wie der Name schon sagt, ist der Speicherzugriff im geschützten Modus geschützt, aber Funktionen wie (peek) und (poke) lassen den Benutzer versuchen, überall hinzuschreiben. Versucht ein OS2XLisp-Anwender an eine Stelle zu schreiben, an der er nichts zu suchen hat, will OS/2 nicht nur das fehlerhafte Programm des Benutzers abbrechen, sondern schickt dem OS2XLisp eine GP-Verletzung (general protection, genereller Schutz).

Dies stellt ein allgemeines Problem für die Programmierung eines Interpreters im geschützten Modus dar. Der geschützte Modus ist großartig, aber die fehlerhafte Anweisung eines Benutzers sollte nur die fehlerhafte Applikation des Anwenders mit einer Fehlermeldung beenden. Das Beenden des Lisp-Interpreters mit einer GP-Verletzung ist nicht zu akzeptieren.

OS2XLisp bietet eine Lösung zu diesem Problem. OS2XLISP.EXE läuft selbst als zu debuggender Prozeß unter DosPTrace. (Der komplette Quellcode in C und Assembler gehört zum Lieferumfang des Pakets. So können Sie, falls es Sie interessiert, die Dinge näher betrachten.) Dies ist die einzige Möglichkeit, GP-Verletzungen unter OS/2 abzufangen.

```
> (poke #xb8000000 666)
break: Segmentation Violation
```

Dies ruft den eingebauten Debugger von OS2XLisp auf, in dem Sie das Problem beseitigen können, oder schauen wo es entstand (wobei es in diesem Fall eindeutig ist). Eine weitere Möglichkeit, den OS2XLisp Debugger aufzurufen, ist das Drücken der Tastenkombinationen [Ctrl][C] oder [Ctrl][Break].

Es gibt aber ein Problem mit DosPTrace. DosPTrace ist normalerweise dem Microsoft CodeView-Debugger vorbehalten. Leider kann immer nur ein DosPTrace aktiv sein. Das bedeutet aber, daß es um OS2XLisp unter CodeView laufenzulassen, oder um mehrere OS2XLisp-Interpreter zur selben Zeit laufenzulassen (vielleicht um mit Semaphoren oder gemeinsamem Speicher zu experimentieren), möglich sein muß, DosPTrace abzuschalten.

```
; SEGS.LSP
; for OS2XLISP
; to examine OS/2 memory segments
; Andrew Schulman 2-August 1988

; requested protection level of seg
(defmacro rpl (x) '(logand ,x 3))

; in LDT or GDT?
(defmacro global? (x) '(zerop (logand ,x 4)))

; does LAR represent accessed segment?
(defmacro accessed? (x) '(= 1 (logand ,x 1)))

; does LAR represent present segment?
(defmacro present? (x) '(= 128 (logand ,x 128)))

; does LAR represent code segment?
(defmacro code? (x) '(= 8 (logand ,x 8)))

; does LAR represent call gate?
(defmacro call-gate? (x) '(= #xE4 ,x))

; does LAR represent readable code?
(defmacro read? (x)
    '(and
        (code? ,x)
        (= 2 (logand ,x 2))))

; does LAR represent writable data?
(defmacro write? (x)
    '(and
        (not (code? ,x))
        (= 2 (logand ,x 2))))

(define (enumsegs func)
    (dotimes
        ; for each possible segment....
        (i #xFFFF)

        ; if there's really a segment there....
        (if (not (zerop (lar i)))

            ; call the callback function,
            ; passing it the segment number,
            ; its access rights (LAR), and
            ; its size (LSL, remembering to
            ; add 1 because LSL is zero-based)
            (apply func (list i (lar i) (1+ (lsl i)))))))

; this is a callback function, passed to (enumsegs)
; by (print-segs) below
(define (show-seg seg seglar segsize)
    (if (< (rpl seg) 3)
        (format stdout "~A\n" seg)
    ; only show details for one of the
    ; four identical segments
        (progn
            (format stdout "~A\t~A\t~A\t"
                ; print segment/selector number
                seg
                ; print whether global (GDT) or local (LDT)
                (if (global? seg)
                    "GDT"
                    "LDT")
                ; print what it is:  data, code,
                ; or call gate
                (cond
                    ((call-gate? seglar)
                        "CALL GATE")
                    ((code? seglar)
                        (if (read? seglar)
                            "READABLE CODE"
                            "EXECUTE-ONLY CODE"))
                    (t
                        (if (write? seglar)
                            "WRITABLE DATA"
                            "READ-ONLY DATA"))))
```

Listing 6: *Der Quellcode von segs.lsp.*

```
            ; if not a call gate, print its size
            (if (not (call-gate? seglar))
                (format stdout "~A  A"
                    segsize
                    (if (= 1 segsize) "byte" "bytes")))
            ; if segment not present in memory, say so
            (if (not (present? seglar))
                (format stdout " (not present)"))
            ; print ruler at the bottom
            (format stdout "\n~A\n" (make-string #\. 70)))))

(define (print-segs)
    (enumsegs #'show-seg))

; non-transparent popup, invoke func inside popup,
; wait for keystroke
(define (popup func)
    (call (getprocaddr viocalls "VIOPOPUP") ^(word 1)
        (word 0))
    (apply func nil)
    (read-char)
    (call (getprocaddr viocalls "VIOENDPOPUP") (word 0)))

; takes optional keyword position-independent parameters,
; e.g., (total-segs :bkgr t) ;
; or (total-segs :bkgr t :pflag nil)
(define (total-segs
        &key (pflag t) (bkgr nil)
        &aux (numsegs 0) (numbytes 0) (gnum 0)
            (lnum 0) (gbytes 0) (lbytes 0))
    ; if running in background, make idle priority class
    (if bkgr
        (call
            (getprocaddr doscalls "DOSSETPRTY")
            (word 0) (word 1) (word 0) (word (getpid))))
    (enumsegs
        ; pass enumsegs a "nameless" function,
        ; created with (lambda)
        (lambda (seg seglar segsize)
            ; only do something for one out of the
            ; four identical segments
            (if (zerop (rpl seg))
                (progn
                    (setf
                        numsegs (1+ numsegs)
                        numbytes (+ numbytes segsize))
                    (if (global? seg)
                        (setf
                            gnum (1+ gnum)
                            gbytes (+ gbytes segsize))
                    ; else if local
                        (setf
                            lnum (1+ lnum)
                            lbytes (+ lbytes segsize)))))))
    ; since the operation takes a long time,
    ; they'll get popup notification when it's done.
    (if bkgr
        (popup
            (lambda ()
                (call
                    (getprocaddr doscalls "DOSSETPRTY")
                    (word 0) (word 2) (word 0)
                    (word (getpid)))
                (format stdout "OS2XLISP finished computing
                            TOTAL-SEGS!\n\n")
                (format stdout "SEGS: ~A\tBYTES: ~A\n\n"
                    numsegs numbytes)
                (format stdout "Press any key to
                            continue\n\n"))))
    ; either print out the results, or return them as a list
    (if pflag
        (not
            (format stdout "Segs:  ~A (GDT ~A, LDT
                    ~A)\tBytes: ~A (GDT ~A, LDT
                    ~A)\n"
                numsegs gnum lnum numbytes gbytes lbytes))
    ; else
        (list numsegs numbytes gnum gbytes lnum lbytes)))
```

Listing 6: *(Ende)*

```
> (print-segs)
12
13
14
15  LDT READ-ONLY DATA  16 bytes
..................................
28
29
30
31  LDT READABLE CODE  3503 bytes
..................................
         .
         .
         .
```

Abbildung 2: *Beispiel in OS2XLisp für die Verwendung von (print-segs).*

OS2XLisp kann mit der Kommandozeilenoption -x aufgerufen werden. Dies schaltet die Verwendung von DosPTrace ab und aktiviert die OS2XLisp-Technik zum Selbstschutz vor GP-Verletzungen durch ein XLisp-Programm. Es verwendet die Befehle LAR, LSL, VERR und VERW des geschützten Modus, um den Zeiger eines Anwenders zu testen, bevor auf den Inhalt zugegriffen wird. LAR liefert die Zugriffsrechte des Segments, LSL liefert die Größe des Segments, VERR teilt mit, ob Leserecht besteht, und VERW teilt mit ob Schreibrecht besteht. Übergeben Sie diesen Befehlen eine ungültige Segmentnummer, so erhalten Sie keine GP-Verletzung.

Der Heap

Ich fand diese Funktionen so hilfreich, daß ich sie auch für Lisp-Programme zugänglich machte.

Listing 6 ist ein Programm namens SEGS.LSP, das durch die von OS2XLisp sichtbaren Segmente geht. Nach dem Laden von SEGS.LSP können Sie die Funktion (total-segs) aufrufen, das die Anzahl der zum OS2XLisp gehörenden Segmente und die Größe ausgibt. Sie können dies einfach im Hintergrund laufenlassen, und wenn Sie (total-segs) anweisen, daß OS2XLisp im Hintergrund läuft, so läuft es mit Idle-Priorität und öffnet ein Fenster, wenn es beendet ist (unglücklicherweise können derzeit noch keine Multithread-Programme unter OS2XLisp geschrieben werden).

Die andere Funktion, die in SEGS.LSP enthalten ist, heißt (printsegs), die eine Funktion an die Funktion (enumsegs) übergibt. Wird diese aufgerufen, gibt sie die Größe des Segments, die Art (lesbare Programmteile, schreibbare Daten, oder was auch immer) und ob das Segment im Speicher ist, aus (*Abbildung 2*).

Warum sind die Segmente in diesem Beispiel zu viert gruppiert? Im geschützten Modus des 286 ist eine Segmentnummer (die Nummer die wir von DosGetInfoSeg erhalten) nicht nur eine Zahl, sondern eine Struktur, die aus einer Anzahl von Feldern besteht, wie Sie in *Tabelle 1* sehen können.

Bitposition	Felddefinition
0,1	Angeforderte Schutzebene (RPL)
2	Tabellenindikator (global/GDT=0, lokal/LDT=1)
3-15	Index in Tabelle

Tabelle 1: *Im geschützten Modus des 286 sind die Segmentnummern Strukturen mit verschiedenen Feldern.*

Die Nummern sagen auch etwas über diese Speichersegmente aus. Der GIS hat ein RPL von 0 und ist natürlich in der globalen Descriptorentabelle (GDT):

```
> (ultoa gis 2) ; binär 96
"1100000"
```

Der LIS hat ein RPL von 3 (11 in binär) und belegt den ersten Platz der lokalen Descriptorentabelle (LDT):

```
> (ultoa lis 2) ; binär 15
"1111"
```

Da die gewünschte Schutzebene in den untersten 2 Bits dieser Struktur ist, welche wir uns als Segmentnummer vorstellen, gibt es vier verschiedene Segmentnummern, die sich nur in ihrer Schutzebene unterscheiden. Das Segment 15 zum Beispiel wird von den Segmenten 12, 13 und 14 überlagert.

Die anderen Segment-Attribute werden im Segment-Zugriffsrecht-Byte geführt, welches wir mit (lar) erhalten. SEGS.LSP definiert verschiedene Makros, um die von (lar) erhaltenen Werte zu interpretieren. Enthält der GIS zum Beispiel schreibbare Daten?

```
> (write? (lar gis))
NIL
> (present? (lar gis))
T
```

Wir können uns die Richtigkeit durch das Prüfen eines Codesegments von OS2XLisp vor Augen führen. Zuerst vergewissern wir uns, daß ein bestimmtes Segment nicht im Speicher ist, indem wir uns die Adresse der Funktion holen, das Segment aus dem Far-Zeiger mit (fp-seg) bilden, die lar des Segmentes lesen und die Funktion (present?) anwenden. Wir rufen diese Funktion auf und Prüfen das Anwesenheitsbit noch einmal:

```
> (present? (lar (fp-seg
    (addr (function sin)))))
NIL
; Sie können vielleicht beim Laden
; das Diskettenlicht sehen
> (sin (/ 3.14159 2))
1
> (present? (lar (fp-seg
    (addr (function sin)))))
T
```

Es funktioniert! Und die Klammern sind auch nicht so schrecklich, wie man am Anfang vermutet, oder?

```
MODULE Dining;

(*
The Five Dining Philosophers in Modula-2
adapted from M. Ben-Ari, PRINCIPLES OF CONCURRENT PROGRAMMING
(Prentice-Hall, 1982), p.113

This module has no operating system dependencies, but depends
upon Sem, which has only been implemented for OS/2
*)

FROM Processes IMPORT StartProcess;

FROM InOut IMPORT WriteString, WriteInt, WriteLn;

FROM Sem IMPORT
    CountingSemaphore, P, V, CSInit,
    BinarySemaphore, Request, Clear, BinInit,
    cobegin, coend, self;

CONST
    NUM_PHILOSOPHERS = 5;
    NUM_FORKS = NUM_PHILOSOPHERS;
VAR
    fork : ARRAY [1..NUM_FORKS] OF BinarySemaphore;
    room : CountingSemaphore;
    fini : CountingSemaphore;
    i : CARDINAL;

MODULE io;
IMPORT
    BinarySemaphore, Request, Clear, BinInit, WriteString,
WriteInt, WriteLn;
EXPORT think, eat;
VAR
    (* iosem has no bearing on problem itself; just for
       printing strings *)
    iosem : BinarySemaphore;

PROCEDURE think (p : CARDINAL) ;
    BEGIN
        Request(iosem);
        WriteString('thinking: '); WriteInt(p,1); WriteLn;
        Clear(iosem);
    END think;

PROCEDURE eat (p : CARDINAL) ;
    BEGIN
        Request(iosem);
        WriteString('eating: '); WriteInt(p,1); WriteLn;
        Clear(iosem);
    END eat;

BEGIN
    (* initialization for internal module *)
    BinInit(iosem);
END io;

PROCEDURE philosopher () ;
    VAR
        i : CARDINAL;
        left, right : CARDINAL;
    BEGIN
        (*
            these don't have to be done each time through
            loop, since each philosopher/process has its own
            stack
        *)
        i := self() - 1;
        left := i;
        right := (i MOD NUM_FORKS) + 1;

        P(fini);

        LOOP
            think(i);
```

Listing 7: *Der Quellcode für das Problem der speisenden Philosophen.*

```
            P(room);
            Request(fork[left]);
            Request(fork[right]);
            eat(i);
            Clear(fork[left]);
            Clear(fork[right]);
            V(room);
        END;
    END philosopher;

BEGIN
    CSInit(room, NUM_FORKS - 1);
    CSInit(fini, NUM_PHILOSOPHERS);

    FOR i := 1 TO NUM_FORKS DO
        BinInit(fork[i]);
    END;

    cobegin();
        FOR i := 1 TO NUM_PHILOSOPHERS DO
            StartProcess(philosopher, 2048);
        END;
    coend();

    P(fini);     (* never cleared! *)
END Dining.

DEFINITION MODULE Sem;

TYPE CountingSemaphore;
PROCEDURE P ['P'] (VAR cs : CountingSemaphore);
PROCEDURE V ['V'] (VAR cs : CountingSemaphore);
PROCEDURE CSInit ['CSINIT'] (VAR cs : CountingSemaphore; count
: CARDINAL);

TYPE BinarySemaphore;
PROCEDURE Request ['REQUEST'] (VAR s : BinarySemaphore);
PROCEDURE Clear ['CLEAR'] (VAR s : BinarySemaphore);
PROCEDURE BinInit ['BININIT'] (VAR s : BinarySemaphore);

PROCEDURE cobegin ['COBEGIN'] () ;
PROCEDURE coend ['COEND'] () ;

PROCEDURE self ['SELF'] () : CARDINAL ;

END Sem.

IMPLEMENTATION MODULE Sem;

(*
contains:
    Counting Semaphores
    Binary Semaphores
    assorted thread-related procedures
*)

FROM SYSTEM IMPORT ADDRESS, ADR;

FROM Storage IMPORT ALLOCATE;

FROM DosCalls IMPORT
    DosSemClear, DosSemSet, DosSemRequest, DosSemWait,
    DosEnterCritSec, DosExitCritSec, DosGetInfoSeg;

TYPE CountingSemaphore = POINTER TO
    RECORD
        cs : LONGINT;
        ms : LONGINT;
        count : CARDINAL;
        countsem : ADDRESS;
        mutexsem : ADDRESS;
    END;

(*
  note:  P() and V() adapted from Kevin Ruddell, "Using OS/2
  Semaphores to Coordinate Concurrent Threads of Execution,"
```

Listing 7: *(Fortsetzung)*

```
 MICROSOFT SYSTEMS JOURNAL, May 1988, Figure 9: "Simulating
  a Counting Semaphore under OS/2," p.26
*)

PROCEDURE P ['P'] (VAR cs : CountingSemaphore);
    VAR blocked : BOOLEAN;
    BEGIN
        blocked := TRUE;
        WHILE blocked DO
            DosSemWait(cs^.countsem, -1);
            DosSemRequest(cs^.mutexsem, -1);
            IF (cs^.count = 0) THEN
                DosSemSet(cs^.countsem);
            ELSE
                DEC(cs^.count);
                blocked := FALSE;
            END;
            DosSemClear(cs^.mutexsem);
        END;
    END P;

PROCEDURE V ['V'] (VAR cs : CountingSemaphore);
    BEGIN
        DosSemRequest(cs^.mutexsem, -1);
        INC(cs^.count);
        DosSemClear(cs^.countsem);
        DosSemClear(cs^.mutexsem);
    END V;

PROCEDURE CSInit ['CSINIT'] (VAR cs : CountingSemaphore; count
: CARDINAL);
    BEGIN
        NEW(cs);
        cs^.cs := 0;
        cs^.ms := 0;
        cs^.count := count;
        cs^.countsem := ADR(cs^.cs);
        cs^.mutexsem := ADR(cs^.ms);
    END CSInit;

TYPE BinarySemaphore = POINTER TO
    RECORD
        s : LONGINT;
        sem : ADDRESS;
    END;

PROCEDURE Request ['REQUEST'] (VAR s : BinarySemaphore);
    BEGIN
        DosSemRequest(s^.sem, -1);
    END Request;

PROCEDURE Clear ['CLEAR'] (VAR s : BinarySemaphore);
    BEGIN
        DosSemClear(s^.sem);
    END Clear;

PROCEDURE BinInit ['BININIT'] (VAR s : BinarySemaphore);
    BEGIN
        NEW(s);
        s^.s := 0;
        s^.sem := ADR(s^.s);
    END BinInit;

(* simulate the cobegin-coend construct *)
PROCEDURE cobegin ['COBEGIN'] ();
    BEGIN
        DosEnterCritSec();
    END cobegin;

PROCEDURE coend ['COEND'] ();
    BEGIN
        DosExitCritSec();
    END coend;

MODULE Self;
IMPORT ADDRESS, DosGetInfoSeg;
EXPORT self;
VAR
    A : ADDRESS;
```

Listing 7: *(Fortsetzung)*

```
    LocalInfo : POINTER TO ARRAY [0..6] OF CARDINAL;
(* return a thread's ID number *)
PROCEDURE self ['SELF'] () : CARDINAL;
    BEGIN
        RETURN LocalInfo^[3];
    END self;

BEGIN
    (* module initialization code *)
    DosGetInfoSeg(A.OFFSET, A.SEGMENT);
    A.OFFSET := 0;
    LocalInfo := A;
END Self;

END Sem.
```

```
; SEMAPH.DEF

LIBRARY SEM INITINSTANCE
DESCRIPTION 'Counting and Binary Semaphores'
DATA MULTIPLE
EXPORTS
    SEM         ; init
    P
    V
    CSINIT
    REQUEST
    CLEAR
    BININIT
    COBEGIN
    COEND
    SELF
```

```
; START.ASM -- auto-init entry point

EXTRN   SEM:FAR

        DOSSEG
        .MODEL large
        .CODE

START   PROC FAR
        call SEM
        mov ax,1     ; return success
        ret
START   ENDP

END     START
```

```
set m2lib=\os2\mod2\lib

m2 sem.def/data:l/thread && m2 sem/data:l/thread
m2 dining/thread

' to make DLL version:
masm start.asm;
link /dosseg start sem junk,sem.dll,,,semaph.def && imblib
sem.lib semaph.def
copy sem.dll \os2\dll        'copy to LIBPATH

link dining,dining,,sem;

' to make non-DLL version:
link dining sem,dining;
```

Listing 7: *(Ende)*

Modula-2

In vielen Dingen paßt Modula-2 gut zu OS/2. So wie OS/2 mit den Ideen des dynamischen Linkens und mehreren Threads entwickelt wurde, basiert Modula-2 auf den Ideen der Module und verschiedener Prozesse.

Einige der Übereinstimmungen von Modula-2 und OS/2 sind in der Tat bemerkenswert. Die Prozedur NEWPROCESS von Modula-2, die eine parallel arbeitende Coroutine mit eigener Umgebung erzeugt, benötigt folgende Parameter:

```
PROCEDURE NEWPROCESS (
(* eine Prozedur ohne Parameter *)
Code : PROC;             (* Stack *)
Workspace : ADRESS;  (* Größe des Stacks *)
WsSize : CARDINAL;
(* erhält das Handle der Coroutine *)
VAR process : ADRESS);
```

Jeder, der mit OS/2 vertraut ist, erkennt, daß dies gleich dem folgenden ist:

```
DosCreateThread (
      void (far *procaddr)
      (void), unsigned far
*thread_id, char far *stack);
```

Im Gegensatz zu C enthält die Sprache Modula-2 Parallelprogrammierung. Niklaus Wirth entwickelte die Sprache Modula-2 als Hochsprache, in der Betriebssysteme, Einheitentreiber und ähnliches geschrieben werden können. Dies verlangt eine Menge Möglichkeiten auf unterster Ebene und Multitasking.

Prozesse sind in Modula-2 selbst definiert, anstatt in Spracherweiterungen (wie etwa Concurrent C, das an den Bell Laboratories entwickelt wurde), was die Portierung gewährleistet. StartProcess von Modula-2 könnte in OS/2 als DosCreateThread, in UNIX als fork, und in DOS durch einen eigenen Scheduler implementiert werden. Wenn aber ein Programm StartProcess statt der darunterliegenden Funktion aufruft, besteht die Möglichkeit, daß es auf andere Umgebungen portierbar ist.

Wirth erkannte auch, daß eine Sprache wie Pascal, bei der eine komplette Neuübersetzung nach jeder Änderung notwendig ist, nicht für größere Projekte geeignet ist. Modula-2 bietet deswegen eine getrennte Übersetzung, was eine Aufteilung des Programms in einzelne Komponenten erlaubt. Es ist aber eine komplette Wartung durch eine modulübergreifende Typprüfung gewährleistet.

Bedenken Sie, daß eine getrennte Übersetzung der abhängigen Module nicht dasselbe ist, wie eine unabhängige Übersetzung. Sogar die Abprüfung des Übersetzungsdatums der abhängigen Module ist in der Sprache definiert. Beachten Sie den Unterschied zwischen diesem Konzept und den Ansätzen der Modularität in C. Der kommende ANSI-Standard sieht Funktionsprototypen als Option vor, und das Vorhandensein der Prototypen in Includedateien ist nur Konvention.

Das Modul

Wie man dem Namen entnehmen kann, ist bei Modula-2 das Modul von zentraler Bedeutung. Sehen wir uns noch einmal das Modula-2-Beispiel in Abbildung 1 an. Dort sehen Sie das Statement IMPORT DosCalls und den Aufruf von DosCalls.DosGetModName. Wenn der Funktionsaufruf wie der Zugriff auf ein Feld in einer Struktur aussieht, so war dies beabsichtigt. Ich hätte auch schreiben können FROM DosCalls IMPORT DosGetModName und dann DosGetModName ohne Zusatzbezeichnung aufrufen können, aber ich wollte zeigen, daß in Modula-2 Module mit Datenstrukturen verwandt sind.

Ein Modul kann Prozeduren, Variablen, Konstanten, Records und Typen exportieren. Sehen Sie sich SEM.DEF in *Listing 7* an. Dort sehen Sie, daß das Modul SEM zwei Typen exportiert, die CountingSemaphore und BinarySemaphore genannt werden, und ebenso verschiedene Operationen auf diese Typen, wie P, V, Request und Clear.

Es ist auch bedeutend, daß ein Modul die Wahl hat, nichts zu exportieren. Ein Modul verbirgt normalerweise Daten und liefert Operatoren um diese Daten zu manipulieren. SEM.DEF exportiert zum Beispiel den Typ CountingSemaphore, um die interne Darstellung in SEM.MOD nicht exportieren zu müssen (SEM.MOD erledigt auch die Speicherallokierung für alle erzeugten Objekte dieses Typs). Das Verbergen von Informationen ist ebenso wichtig, wie das Preisgeben von Informationen. Das Ziel ist es, die Verbindungen zwischen den Modulen möglichst gering zu halten. Diese Modularisierung spielt auch später eine Rolle, wenn Sie versuchen, eine Anwendung in verschiedene dynamische Linkbibliotheken aufzuteilen.

Es gibt zwei wichtige Komponenten: Definitionsmodule und Impementierungsmodule. Ein Definitionsmodul enthält die Schnittstellenbeschreibung. Diese .DEF-Datei wird vorübersetzt in eine Symboltabelle (.SYM-Datei), welche zum Prüfen der Intermodulreferenzen benötigt wird. Da ein Modul nur den Definitionsteil eines anderen Moduls importieren kann, braucht man nur den Definitionsteil zu schreiben. Die Definitionsmodule können als »Sprache« benutzt werden, um Funktionsspezifikationen zu schreiben.

Das Implementierungsmodul enthält natürlich das Programm. Analog zu dem, was ich früher gesagt habe, ist dies der beste Platz für alle betriebssystemabhängigen Funktionen. Ein Implementierungs-Modul kann auch einen automatischen Initialisierungsteil enthalten, was vergleichbar der Option INITINSTANCE der OS/2-DLLs ist. Ein Modul kann auch Untermodule besitzen, wie Self in SEM.MOD (Listing 7). Zu guter Letzt ist das Implementierungsmodul auch der Platz, um die Details des gegenseitigen Ausschlusses für beschränkte Ressourcen, wie Bildschirm oder Tastatur zu verbergen.

Da Modula-2 eingebautes Multiprogramming besitzt, sind seine Bibliotheken für den Aufruf aus mehreren Threads ausgelegt. Daher ist es ein Leichtes, Applikationen

mit mehreren Threads und dynamische Linkbibliotheken unter OS/2 und Windows in Modula-2 zu programmieren. Bis zum Erscheinen der MTDYNA-(Multithread)-Bibliotheken in C 5.1 mußte Microsoft eine Liste der Bibliotheksfunktionen veröffentlichen, die nicht von Anwendungen mit mehreren Threads, Windows oder OS/2 dynamischen Linkbibliotheken aufgerufen werden können.

Semaphore

Modula-2 bietet keine Semaphor- oder Signalfunktionen, die besser sind, als diejenigen in OS/2. In der Stony Brook-Implementierung von Modula-2 arbeitet das Processes Modul, wie es in Niklaus Wirth's *Programming in Modula-2* (Springer Verlag 1983) beschrieben ist, mit Threads und Semaphoren. Das Modul Processes definiert den Typ Signal und die Operationen Send, Wait, Init, Awaited und StartProcess. Die Dokumentation von Stony Brooks rät, Signale nur zu verwenden, wenn Sie auf Kompatibilität mit anderen Versionen von Modula-2 Wert legen, da OS/2 effektivere Methoden der Synchronisation bietet.

Obwohl es leicht ist, Programme mit vielen Threads mit der Prozedur StartProcess kompatibel zu schreiben, bietet Modula-2 nicht viel für die Prozeß-Synchronisation. Da es in Modula-2 nur die Funktionen Send und Wait gibt, sollten wir über eine Synchronisation auf einer höheren Ebene, wie sie von OS/2 mit DosSemRequest geboten wird, nachdenken. Es gibt eine große Lücke zwischen den mächtigen Semaphoren, wie sie OS/2 bietet, und der klassischen Parallelverarbeitungs-Theorie. Modula-2 ermöglicht uns, die Lücke zu überbrücken.

Die speisenden Philosophen

In Listing 7 habe ich in Stony Brooks Modula-2 das Problem der fünf speisenden Philosophen implementiert, das vom Holländischen Computer-Wissenschaftler E. W. Dijkstra 1965 gestellt wurde und das ein gutes Beispiel der Multiprogrammierung darstellt. Das folgende ist eine Zusammenfassung aus M. Ben-Aris *Principles of Concurrent Programming* (Prentice-Hall, 1982) auf Seiten 109-110, von dem auch unsere Lösung geborgt ist.

»Das Problem spielt in einem Kloster, wo die fünf Mönche Philosophen sind. Jeder Philosoph wäre glücklich, wenn er nur denken bräuchte, ohne daß er essen müßte. Jedoch das Leben eines Philosophen ist ein endloser Kreis: *wiederhole* denken, essen *ewig*... In der Mitte des Tisches steht eine Schüssel Spaghetti, die immer wieder aufgefüllt wird. Es gibt 5 Teller und 5 Gabeln. Will ein Philosoph essen, so betritt er den Speiseraum, setzt sich, ißt und kehrt dann zu seiner Zelle zum Denken zurück. Aber die Spaghetti sind so heillos ineinander verschlungen, daß 2 Gabeln zum Essen notwendig sind. Das Problem besteht darin, ein Ritual (Protokoll) zu entwickeln, das den Philosophen das Essen ermöglicht. Jeder Philosoph darf nur die beiden Gabeln neben seinem Teller verwenden. Das Protokoll muß die Anforderungen erfüllen: gegenseitiger Ausschluß (keine zwei Philosophen dürfen gleichzeitig auf dieselbe Gabel zugreifen) und frei von Deadlocks und Ausschluß.«

Die fünf Philosophen stellen fünf Prozesse (oder in OS/2 Threads) dar. Die fünf Gabeln stellen kritische Ressourcen dar, für die fünf Semaphore den gegenseitigen Ausschluß gewährleisten. Da fünf Philosophen (Threads) und fünf Gabeln (Ressourcen) vorhanden sind, und jeder Philosoph zwei Gabeln benötigt, können immer zwei Philosophen gleichzeitig essen. Die Gabeln können nicht durch einen einzigen kritischen Bereich geschützt werden, denn sonst könnte immer nur ein Philosoph essen.

Da alle fünf Philosophen dasselbe tun, führen alle fünf Prozesse dasselbe Stück Programm aus, die Prozedur philosopher in Listing 7. Da sowohl OS/2 als auch Modula-2 gleichzeitige Prozesse als parameterlose Prozeduren behandeln (und da die Philosophen wissen müssen, wer sie sind), holt sich jeder Thread beim Ausführen der Prozedur philosopher mit der Funktion self seine Identifikation. In SEM.MOS ist die Funktion self in der Art implementiert, daß die Thread ID-Nummer aus dem lokalen Info-Segment geholt wird.

Das Leben ist ein endloser Kreislauf, daher besteht die Prozedur philosopher aus einer Endlosschleife. Für jeden Philosophen bleiben aber einige Gegebenheiten gleich, die Identität und die Gabeln, die er benutzen soll. Sie werden einmal zugeteilt und im Speicher des Philosophen abgelegt (welcher sich auf dem Stack des Threads befindet).

Im Inneren der Schleife sitzt jeder Philosoph in seiner Zelle und denkt eine Zeitlang. Danach versucht er den Speiseraum zu betreten. Wenn es fünf Philosophen gibt (die Konstante NUM_PHILOSOPHERS kann geändert werden), dann dürfen vier Philosophen im Raum sein. Philosopher führt den Code P(room) aus. Room ist ein Zählsemaphor, und P ist die Warteoperation (das holländische Wort für versuchen ist »proberen«). Ist der Philosoph zur Rückkehr in die Zelle bereit, so führt er den Code V(room) aus. V ist die Freigabeoperation (das holländische Wort für freigeben ist »vrygeven«).

Die Implementierung der Zählsemaphore ist im Modul SEM.MOD enthalten. Es beruht stark auf der OS/2-Simulation der Zählsemaphore aus *Koordination von Threads mit Semaphoren* (MSJ, 2. Jg./Heft 4).

Im Modul DINING.MOD ist kein OS/2-spezifischer Code enthalten. Der ganze systemspezifische Code befindet sich im Modul SEM.MOD (und in den Bibliotheksmodulen von Stony Brook Software. Der ganze Quellcode der Bibliothek ist im Lieferumfang enthalten). Dies ist einfach ein OS/2-Programm mit Threads und Semaphoren.

Im Speiseraum wartet der Philosoph bis er seine linke Gabel nehmen kann, und anschließend auf seine rechte Gabel. Deadlock stellt hier kein Problem dar. Das Zählsemaphor room hält die Philosophen, die versuchen, Gabeln zu nehmen, kleiner als die Anzahl der Gabeln. Daher kann immer ein Philosoph essen. Ein Array von binären Semaphoren stellt die Gabeln dar. (Binäre

Semaphore sind einfacher als Zählsemaphore und können durch einen einfachen Aufruf von DosSemRequest sowie DosSemClear implementiert werden.) Nach dem Essen gibt der Philosoph die Gabeln frei, verläßt den Raum und beginnt von Neuem.

Das Starten der fünf Philosophen in der Initialisierung im Modul DINING.MOD sieht folgendermaßen aus:

```
FOR i := 1 TO
NUM_PHILOSOPHERS DO
    StartProcess(philosopher, 2048);
END;
```

Um sicherzustellen, daß alle Philosophen zur selben Zeit starten, simulieren wir das Sprachkonstrukt Cobegin-Coend. In SEM.MOD sind diese als einfache Funktionsaufrufe von DosEnterCritSec und DosExitCritSec implementiert. Solange der Hauptprozeß in seiner kritischen Phase ist, kann er Prozesse starten, aber diese können nicht laufen. Sobald er seine kritische Phase beendet, starten alle Prozesse zusammen.

Ich hatte ein Problem, als ich DINING.MOD programmierte. Alle Threads starteten richtig, aber aus irgendeinem Grund beendete sich das Programm anschließend. Das Problem war, daß der Hauptthread, nachdem er alle Philosophen zum Leben erweckt hatte, seine Aufgabe verrichtet hatte, und sich deswegen beendete. Ich führte ein anderes Semaphor namens fini ein. Der Hauptthread wartet auf dieses Semaphor, aber keiner der Philosophen gibt es je frei. Ich fürchte, daß jemand sagen könnte, dies bedeutet, daß das Philosophieren immer weiter geht, und kein Ende nimmt. Um den ansonsten endlosen Kreislauf des Essens und des Philosophierens zu beenden, müssen Sie [Ctrl][C] drücken.

Die Philosophen zeigen an, was sie tun, indem sie kleine Ausgaben auf dem Bildschirm machen:

```
thinking: 4
eating: 2
eating: 1
eating: 4
thinking: 5
thinking: 3
thinking: 2
thinking: 1
eating: 3
        .
        .
        .
```

Um diese Ausgaben am Bildschirm zu trennen, müssen wir ein weiteres Semaphor iosem benutzen. Da dies nichts mit dem Philosophenproblem zu tun hat, ist der gesamte Programmteil in einem internen Modul von DINING.MOD untergebracht.

Ein weiteres Problem ergab sich beim Programmieren von DINING.MOD. Das Programm wurde mit einer GP-Verletzung beendet. Nachdem ich einige Zeit verbracht hatte, um die OS/2-Registeranzeige zu entziffern, erinnerte ich mich, daß ein Vorteil von Modula-2 die Fehlerprüfung zur Laufzeit ist. Ich aktivierte den Compiler-Schalter /CHECK und übersetzte neu. Beim nächsten Programmlauf erhielt ich eine Fehlermeldung über eine Arraygrenzenverletzung in der Zeile 79 von DINING.MOD. Dies war auch die fehlerhafte Stelle. Dies allein ist es Wert, Modula-2 in Betracht zu ziehen.

Nachdem das Programm lief, war der nächste Schritt die Aufspaltung von SEM.MOD als eine dynamische Linkbibliothek, SEM.DLL. Dies verlangt die Erstellung einer DEF-Datei für den Linker, die aber nicht mit der ebenso im Listing 7 gezeigten Modula-2 DEF-Datei zu tun hat (M2 heißt der Stony Brook Modula-2-Compiler). Zusätzlich zu mehreren Kopien von DINING.EXE, die gleichzeitig laufen, werden diese mit SEM.DLL dynamisch gelinkt. Ich rief SEM.DLL auch aus C auf (bedenken Sie, daß EM.MOD selbst Speicher für die Zählsemaphore allokiert):

```
typedef void
    far *CountingSemaphore;
            .
            .
            .
CountingSemaphore room;
/* Zeiger auf Zeiger */
CSInit(&room, 4);
            .
            .
            .
P(room)
```

Stony Brook Modula-2 kommt mit Möglichkeiten zum Anschluß an andere Sprachen, die meines Wissens einzigartig sind. Die meisten C-Compiler geben Ihnen die einfache Möglichkeit, zwischen der C- und der Pascal-Aufrufkonvention zu unterscheiden, Stony Brook Modula-2 hat die optionalen Schlüsselworte REVERSED (ablegen der Parameter in umgekehrter Reihenfolge), LEAVES (Prozedur läßt die Parameter auf dem Stack), ALTERS (geben Sie das Register an, das die Prozedur ändert), RETURNS (geben Sie das Register an, in dem die Funktion das Ergebnis zurückliefert) und VARIABLE (die Prozedur erlaubt eine variable Anzahl von Argumenten).

Zum Importieren der C-Funktion printf in Modula-2 brauchen Sie die Schlüsselworte REVERSED, LEAVES, VARIABLE und ALTERS(AX, BX, CX, DX, ES). Der Hauptvorteil ist aber, daß Sie LEAVES angeben können, ohne gleichzeitig REVERSED angeben zu müssen. Dies erlaubt es, eine Funktion zu schreiben, die Daten auf dem Stack für eine andere Funktion ablegt.

Stony Brook Modula-2 paßt wunderbar zu C, was ungewöhnlich für ein Produkt ist, in dessen Anzeigen mit dem Spruch geworben wird »Ich komme, um C zu besiegen, nicht um es zu loben«.

```
declare sub DosGetInfoSeg (seg arg1%, seg arg2%)
DosGetInfoSeg gis%, lis%
def seg = gis%
print "Uhrzeit: "; peek(8);":";peek(9);":";peek(10)
```

Die OS/2-Befehlszeile zum Kompilieren, Linken und Starten von INFOSEG BAS lautet:

```
C>bc infoseg.bas; && link infoseg; && infoseg
```

Listing 8: *Erzeugen einer Basic-Utility.*

Basic

Basic hat sich seit der Entwicklung der Sprache 1964 durch John Kemeny und Tom Kurtz stark weiterentwickelt. Sie brauchen keine Zeilennummern mehr, und es existieren Funktionen und Unterprogramme. Auch ist es ein Zeichen der Veränderung, daß Microsoft Basic 6.0 ein Compiler ist (kein interaktiver Interpreter), der eine breite Palette von befehlszeilenähnlichen Kommandoschaltern bietet, wie /Zi, /ah und /d/e/x.

Auf eine Art paßt Basic gut zu OS/2. Die Basic-Tradition bei früheren Microsoft Compilern war es schon immer, Programme zu erzeugen, die eine Laufzeitumgebung benötigen - BRUN.COM. In OS/2 bietet es sich an, diese Laufzeitumgebung als dynamische Linkbibliothek auszulegen (BRUN60EP.DLL). Vergleichen Sie dies mit C, welches komplette Applikationen bildet.

CRTLIB.DLL ist eine wenig bekannt Option in C 5.1. Sogar die Laufzeitumgebung des Basic 6.0 für den Real Mode scheint geändert worden zu sein, um der DLL ähnlich zu sein.

In jedem Fall ist Basic immer noch die Sprache für Beginner. Aber speziell für diesen Zweck bietet Basic großartige Errungenschaften. Der Anfänger wird damit überfordert, daß Speicher für Strings allokiert werden muß. Deshalb erledigt Basic dies selbst:

```
print "hello " + "world"
```

Dies ist vergleichbar mit

```
puts(strcat("hello ", "world"));
```

was Sie in der Regel nicht tun dürfen, da sonst "world" Speicher überschreibt, der ihm nicht gehört. Im Gegensatz hierzu übernimmt Basic die volle Kontrolle bei der Verwaltung des Speichers für Strings. Daher sind Strings in Basic einfach zu handhaben, intuitiv und deshalb mächtig.

Bedenken Sie auch, daß eine Zeile in Basic ein komplettes Programm sein kann. Die Äquivalente in Modula-2 und C benötigen mehrere Zeilen (oder sogar 100 wenn Sie unter Windows programmieren). Komplexe Programme, die in C leicht programmiert werden können, sind andererseits schwierig in Basic zu schreiben.

```
' enumproc.bas
' bc /e enumproc; && link enumproc; && enumproc brun60ep

if command$ = "" then
    dll$ = "BRUN60EP.DLL"
elseif instr(command$, ".") then
    dll$ = command$
else
    dll$ = command$ + ".DLL"
endif

pipe = freefile

' makes it easy to detect end of pipe's output
on error goto closefile

foundit = 0

open "pipe:exehdr " + dll$ + " 2>nul" as pipe

foundfile = 1       ' found EXEHDR

' get rid of initial EXEHDR output
do while instr(buf$, "Exports:") = 0
    line input #pipe, buf$
loop
line input #pipe, buf$

foundfile = 2       ' found named DLL

print "Routines exported by "; dll$

do while 1
    line input #pipe, buf$
    buf$ = mid$(buf$,16)
    procname$ = left$(buf$, instr(buf$, " ") - 1)
    print procname$
loop

closefile:
    if foundfile = 0 then
        print "Can't find EXEHDR"
    elseif foundfile = 1 then
        print "Can't find "; dll$
    endif
    close pipe
    system
```

Listing 9: *ENUMPROC.BAS ist ein Programm, das alle von einer DLL exportierten Funktionen ausgibt.*

Flexible Syntax

Der PRINT-Befehl ist ein anderes Beispiel. Er verfügt über viele Optionen, aber wenn Sie nur PRINT angeben, erhalten Sie schon ein zufriedenstellendes Ergebnis. Das Ausdrucken der Modulnummer und des Modulnamens in Abbildung 1 ist in Basic kürzer als in allen anderen Sprachen.

Das Basic-Beispiel in Abbildung 1 zeigt ebenso, daß in der Regel Variablen automatisch beim ersten Erscheinen deklariert werden. Das erste Auftauchen der Variablen i% bedeutet, daß es sich um eine Integervariable handelt. buf$ ist eine String-Variable, da der Name am Ende ein Dollarzeichen hat. Das Symbol am Ende des Namens wird als Typbezeichnung verwendet.

Basic-Strings sind keine ASCIIZ-Strings. Wenn Sie von OS/2 einen String erhalten, müssen Sie wissen, wie Sie diesen handhaben können. (In Abbildung 1 habe ich rtrim$ verwendet, um die hinteren Blanks abzuschneiden.)

```
' peeker.bas
' compile with bc peeker.bas /d/e/x; && link peeker;

on error goto errorhandler

do while 1
    input; "seg"; sgm%
    print chr$(9);
    input; "offset"; ofs%
    print chr$(9);

    def seg = sgm%
    print peek(ofs%)
loop

errorhandler:
    if err = 70 then
        print "Permission denied"
        resume next
    else
        print "Unknown error #"; err
        system
    endif
```

Listing 10: *Der Quellcode von PEEKER.BAS.*

Auf der anderen Seite stellt das Erzeugen von ASCIIZ-Strings kein Problem dar. Sie brauchen am Ende nur einen 0-Delimiter anzufügen:

```
asciiz$ = "hello world" + chr$(0)
```

Durch die Möglichkeiten der Stringzuweisung und der Stringverknüpfung muß Basic in der Lage sein, Strings im Speicher zu verschieben. Dies stellt normalerweise kein Problem dar. Benötigen Sie aber die Adresse eines solchen variabel langen Strings, so sollten Sie diese am besten sofort verwenden. In Abbildung 1 erfordert das Holen der Adresse von buf$ zwei Schritte. Zuerst muß das Segment mit varseg geholt werden, dann holen wir die Adresse des Strings mit sadd.

Das ist ein gutes Beispiel für die Programmierung in Basic. Einfache Aufgaben sind auch einfach zu lösen, aber schwierigere Dinge sind kompliziert zu erledigen. Es ist aber wichtig, zumindest eine Sprache zu haben, die dies bietet.

Nichts zu deklarieren?

Fortgeschrittene Basic-Programmierer verwenden die DECLARE-Anweisung von Microsoft Basic, die der Schlüssel zu anderen Sprachen wie C, Assembler und Modula-2 ist, aber auch zu den dynamischen Linkbibliotheken und zu OS/2.

Abbildung 1 zeigt eine typische Deklaration aus den Headerdateien, die von Microsoft mitgeliefert werden. Einmal deklariert, kann die Funktion DosGetModName wie jede andere Funktion verwendet werden. Das Schlüsselwort BYVAL in der Deklaration ist sehr wichtig. Standardmäßig übergibt Basic den Offset (Near-Adresse) einer Variablen. BYVAL aber veranlaßt die Übergabe by Value.

```
' smsw.bas
' compile with bc smsw.bas; && link smsw;

declare sub DosCreateCSAlias (byval p1%, seg p2%)

code$ =
    chr$(&hc8) + chr$(0) + _
    chr$(0) + chr$(0) +           ' enter 0,0
    chr$(&h8b) + chr$(&h5e) + _
    chr$(&h06) +                  ' mov bx,word ptr [bp+6]
    chr$(&h0f) + chr$(&h01) + _
    chr$(&h27) +                  ' smsw word ptr [bx]
    chr$(&hc9) +                  ' leave
    chr$(&hca) + chr$(&h02) + chr$(&h00) ' retf 2

DosCreateCSAlias varseg(code$), csalias%
def seg = csalias%
call absolute(x%, sadd(code$))

if x% and 2 then
    print "Coprocessor present"
elseif x% and 4 then
    print "No coprocessor -- emulate"
else
    print "Something is very wrong"
endif
```

Listing 11: *Der Quellcode von SMSW.BAS.*

Ein anderes Schlüsselwort ist SEG. Es teilt Basic mit, einen Aufruf by Reference auszuführen, aber eine vollständige 4-Byte-Adresse mit Segment zu verwenden. Ist einmal ein Argument in dieser Weise deklariert, brauchen Sie sich keine Sorgen mehr um seine Adresse zu machen. Sehen Sie hierzu auch das kleine Beispielprogramm INFOSEG.BAS in *Listing 8*.

Hier sind einige Dinge zu bemerken. Da wir gewöhnlicherweise alle Fehlercodes von DosGetInfoSeg ignorieren, können wir DECLARE verwenden, um sie als Unterprogramm und nicht als Prozedur zu deklarieren. Da es ein Unterprogramm ist, benötigen wir keine Klammerung der Argumente beim Aufruf. Durch das Schlüsselwort SEG verwendet Basic beim Aufruf die Far-Adresse der Variablen. Wir brauchen diese Variablen nicht zu deklarieren, sondern nur zu benützen (was für eine gute Idee). Alles in allem sieht so ein Programm recht natürlich aus.

Durch die Deklaration von DosGetInfoSeg als Unterprogramm und den Aufruf ohne Klammern haben wir in Wirklichkeit ein neues Basic-Schlüsselwort gebildet.

Eine weitere nützliche OS/2-Funktion, in der wir einen Fehlercode ignorieren können, ist DosBeep, eine sehr nützliche Funktion, da die Statements SOUND und PLAY im geschützten Modus von Basic nicht mehr vorhanden sind.

```
declare sub Honk alias "DOSBEEP"_
  (byval freq%, byval dur%)
Honk 880, 100
```

Und wieder haben wir etwas geschaffen, das wie ein eingebautes Basic-Schlüsselwort arbeitet. Wegen der ALIAS-Option von DECLARE können wir DosBeep linken, es aber ganz anders nennen.

Ein Gefühl von »Makro-Basic«

Das Statement OPEN ist typisch für Basic. Es besitzt eine Menge Parameter, die optional sind, und eine brauchbare Voreinstellung, wenn Sie die Optionen nicht benutzen.

```
open "foo.bar" as #1
```

öffnet die Datei zum wahlfreien Lese-/Schreibzugriff. Das ist für Anfänger und einfache Applikationen brauchbar. Aber Sie können auch programmieren:

```
open "foo.bar" for output access write_
lock write as #1
```

Der String, der den Dateinamen angibt, kann aber auch Parameter beherbergen, die dem OPEN eine andere Bedeutung geben:

```
open "com1:9600,n,8,1,bin" as #1
```

Eine Neuerung enthält die Version des Basic für den geschützten Modus. OPEN "PIPE:" erleichtert den Gebrauch der OS/2 Pipe-Möglichkeiten, wodurch OS/2-Programme über Standard-Dateiaufrufe kommunizieren können:

```
open "pipe:exehdr brun60ep.dll 2>nul"_
as pipe
```

ENUMPROC.BAS in *Listing 9* gibt die Namen aller exportierter Funktionen einer DLL in OS/2 aus.

```
C>enumproc \dll\brun60ep
```

gibt alle Funktionen der Basic-Laufzeitbibliothek aus (die Namen wie B$DRAW und B$POKE haben). ENUMPROC.BAS weiß nichts über die Struktur einer DLL-Datei, aber es weiß, daß EXEHDR.EXE sich mit diesen Dateien auskennt und verarbeitet daher die Ausgaben von EXEHDR. Da die Ausgaben von EXEHDR in eine Basic-Variable gelangen, können wir sie in der Art und Weise manipulieren, wie wir wollen. Daher gestattet es das Statement OPEN "PIPE:" einem Programm, ein anderes als Unterprogramm zu benutzen.

In ENUMPROC.BAS verwenden wir auch die Umleitung von OS/2 ("2>nul"), um ein paar Zeilen, die EXEHDR an stderr sendet, auszublenden, ebenso wie die wichtige Fähigkeit, die Basic besitzt, um das Ende der Eingabe von EXEHDR zu erkennen. Obwohl dies oftmals beim Gebrauch von Pipes in OS/2 schwierig herauszufinden ist (was zu hängenden Programmen führt), brauchen wir in Basic nur zu schreiben:

```
on error goto closefile
```

(Das zeigt, daß Basic oftmals wie Pseudocode aussehen kann.) Wir erhalten die Eingaben von EXEHDR in einer Endlosschleife (do while 1), aber wenn unser Versuch, Eingaben von EXEHDR zu erhalten, fehlschlägt, erhalten wir einen Fehler. Da wir eine Fehlerbehandlung installiert haben, wird unsere Fehlerfunktion (closefile) aufgerufen, und wir können beenden.

Die drei E's

Eine großartige Fähigkeit von Basic ist die Behandlung der drei E's: Error, Exception und Event. Möglicherweise sind diese ON-Statements von PL/1 abgeleitet. Die in Microsoft Basic erlauben Ihnen, eine Menge von Laufzeitfehlern, asynchronen Ereignissen und Signalen zu behandeln.

Im *Listing 9* ist keine Stelle vorhanden, an der wir gezielt ein Ausgabeende von EXEHDR abprüfen. Basic erledigt alle Prüfungen im Hintergrund - aber nicht als Hintergrundthread. Der Compiler setzt Funktionsaufrufe einer Fehlerprüfroutine ein. Das macht die Programme verständlicher, da die Behandlung der Ereignisse nicht durch viele IF-Abfragen unübersichtlich wird. Programmteile, die Fehlerbehandlungsfunktionen verwenden, sind in der Regel leichter verständlich als defensiv programmierte.

Stony Brook Modula-2 besitzt eine ähnliche Möglichkeit in seinem Fehlermodul, und in C kann etwas ähnliches durch die Verwendung von setjmp/longjmp implementiert werden. Aber das Basic-Statement ON stellte die am leichtesten anzuwendende Fehlerbehandlung dar, und dieselbe Technik wird zum Bearbeiten von Nichtfehler-Bedingungen verwendet, wie zum Beispiel das Setzen eines Timers (ON TIMER), das Abfragen einer Tastatureingabe (ON KEY), das Ankommen von Daten an einer seriellen Schnittstelle (ON COM), oder im geschützten Modus das Empfangen eines Signals (ON SIGNAL).

Das Programm PEEKER.BAS im *Listing 10* zeigt einen weiteren interessanten Aspekt der Fehlerprüfung im geschützten Modus von Basic. Wie Sie aus der Diskussion von OS2XLisp wissen, stellt das Verwenden von PEEK- und POKE-Befehlen im geschützten Modus von OS/2 eine Gefahr dar. PEEKER.BAS läuft in einer Endlosschleife, und fragt den Benutzer nach Segment und Offset. Kann das Byte an der Stelle Segment:Offset gelesen werden, so wird der Wert ausgegeben. Ansonsten fängt Basic die tödliche GP-Verletzung ab und übergibt sie an die Fehlerbehandlung, die »Permission denied« ausgibt. Um einen unerlaubten Speicherzugriff im geschützten Modus zu kennzeichnen, verwendet Basic ebenso die Fehlernummer 70, die es für einen unerlaubten Dateizugriff verwendet. Es macht Sinn, Speicher ebenso als gemeinsam benutzbare, aber zeitweise geschützte Ressource, wie eine Datei zu behandeln.

Wenn Sie PEEKER.EXE unter CodeView laufen lassen, so sehen Sie, daß Basic die Anweisungen VERR, VERW und LSL bei der Ausführung von PEEK verwendet. Der Fehler wird also vermieden, anstatt die Folgen abgefangen.

Die Portierung nach OS/2

Nicht nur PEEK und POKE, sondern ein Großteil des als maschinenabhängig bezeichneten Codes ist tatsächlich abhängig vom Betriebssystem. Die Handbücher von Microsoft Basic 6.0 und QuickBasic 4.0 belegen dies. Die Aussage »Dieses Programm enthält hardwareabhängige Anweisungen. Es arbeitet nur auf IBM PC/XT und PC/AT kompatiblen Computer richtig« ist nicht richtig. Tatsächlich arbeiten solche Programme unter MS-DOS korrekt, aber nicht auf einem IBM AT unter OS/2.

```
' smsw2.bas
'
' To compile and link (all on the same line):
' bc smsw2.bas; && masm protmode.asm; && link smsw2
'   protmode, smsw2;

declare sub SMSW alias "_SMSW" (x%)

smsw x%
if x% and 2 then
    print "Coprocessor present"
elseif x% and 4 then
    print "Using emulator"
else
    print "Something is wrong!"
endif
```

```
; protmode.asm
; requires MASM 5.1

.model medium,basic
.286p
.code

_smsw   proc far    arg1:near ptr word
        mov bx,arg1
        smsw word ptr [bx]
        ret
_smsw   endp

_lsl    proc far    segm:word
        sub dx,dx
        sub ax,ax
        lsl ax,segm
        ret
_lsl    endp

end
```

Listing 12: *Der Quellcode von SMSW2.BAS und PROTMODE.ASM.*

Das Handbuch zeigt im Kapitel von DEF SEG, wie Sie DEF SEG, PEEK und POKE verwenden können, um die Taste [CapsLock] zu manipulieren. Dieses Programm, das in fremden Speicherbereichen schreibt und aus ihnen liest, arbeitet unter OS/2 nicht. Unter OS/2 sollten Sie für die Lösung dieses Problems die Funktionen DosDevIOCtl und KbdSetStatus verwenden.

Das Handbuch im Kapitel CALL ABSOLUTE zeigte die Verwendung des Interrupts 11 Hex, um zu erkennen, ob ein mathematischer Coprozessor vorhanden ist. Die beste Möglichkeit unter OS/2 ist die Funktion DosDevConfig, die sehr ähnlich der Ausstattungsliste des ROM BIOS-Interrupts 11H ist.

Da jedoch der Sinn dieses Programms die Demonstration der Ausführung eines Maschinenprogramms ist, das sich in einer Basic-Variablen befindet, können wir dies auch unter OS/2 tun.

Im Programm SMSW.BAS in *Listing 11* wird ein Maschinenprogramm in einen Basic-String übertragen und ausgeführt. Der ausführbare String code$ enthält die Anweisung SMSW des geschützten Modus, die das Statuswort des 286 liest. Ist Bit 1 dieses Wortes gesetzt, ist ein Coprozessor vorhanden. Ist Bit 2 gesetzt, ist kein Coprozessor vorhanden, und/oder es ist ein INT 7 Interrupthandler installiert, um den Coprozessor per Software zu emulieren.

Bevor wir den String mit CALL ABSOLUTE ausführen, müssen wir bedenken, daß wir im geschützten Modus den Datensegment-Selektor nicht in das Register CS laden dürfen. Daher rief ich im Programm SMSW.BAS die OS/2-Funktion DosCreateCSAlias auf, die einen Datensegment-Selektor nimmt und einen Selektor zu diesem Segment liefert, der ausführbar ist.

Dies ist aber bei der Verwendung von CALL ABSOLUTE in Basic unter OS/2 nicht notwendig. Wenn Sie PEEKER.EXE unter CodeView laufenlassen, sehen Sie, daß der Compiler automatisch den Aufruf von DosCreateCSAlias durchführt. Da dies nicht dokumentiert ist, sollten Sie aber den Aufruf von DosCreateCSAlias dennoch durchführen. In diesem Fall ergibt sich beim zweiten Aufruf ein Fehler, da Sie kein CS-Alias für etwas erhalten können, das bereits ein Codesegment ist.

Dies ist noch ein guter Augenblick, um herauszustellen, daß CodeView Basic kennt, was nicht nur zum Fehlersuchen nützlich ist, sondern auch um sich anzusehen, welchen Code der Compiler erzeugt. CodeView besitzt einen Basic-ähnlichen Ausdrucksauswerter, der Abfragen wie

```
? varseg(code$)
&H002f
```

beantwortet.

Basic-Erweiterungen

Es gibt eine bessere Möglichkeit, Basic zu erweitern, als Maschinenprogramme in Variablen zu packen. Da der Microsoft Basic Compiler Standard-.OBJ-Dateien erzeugt, und wegen der wichtigen DECLARE-Anweisung, können wir alle Funktionen aus Standard-.OBJ-Dateien linken. *Listing 12* zeigt sowohl das Programm für PROTMODE.ASM und SMSW2.BAS, und zeigt auch eine bessere Lösung, um den Befehl SMSW in Basic zu übernehmen.

Beachten Sie, wie es mit Basics DECLARE SUB möglich ist, das natürlicher aussehende SMSW% zu erzeugen, anstatt x% = SMSW zu schreiben. Da MASM 5.1 Möglichkeiten für die Unterstützung von Hochsprachen bietet, können wir nach der Direktive

```
.MODEL MEDIUM,Basic
```

MASM die Verantwortung für die Erzeugung von Basic-kompatiblem Code übertragen.

Da Sie jetzt wissen, daß wir die Anweisung SMSW leicht in Basic übernehmen können, werden Sie sicher nicht überrascht sein, daß wir ebenso leicht unsere anderen Freunde LSL, LAR, VERR und VERW in das Repertoire von Basic übernehmen können. PROTMODE.ASM enthält auch die Befehle für die Funktion LSL. Der Gebrauch von Basic sieht folgendermaßen aus:

```
declare function LSL& _
    alias "_LSL" (byval x%)
print "LSL for some Basic _
    data segment is"; _
    LSL(varseg(x%))
```

Da die Integer-Variablen in Basic immer mit Vorzeichen behaftet sind, würden wir immer, wenn der von der Funktion LSL zurückgegebene Wert größer als 32767 ist, negative Zahlen sehen. Deshalb gibt die Basic-Funktion LSL einen 4-Byte-Wert zurück, was an dem & erkennbar ist.

Was ist neu, was fehlt?

Etliches ist im OS/2-Basic nicht mehr vorhanden. Die Basic-Funktionen und -Statements, die Zugriffe auf Ein-/Ausgabebausteine zuließen, fehlen. (Wir können diese ebenso in Assembler implementieren, aber wir benötigen ein IOPL-Statement aus der Datei OS/2.DEF). Die Basic SOUND-Statements fehlen auch (wir haben schon gesehen, daß DosBeep dies übernehmen kann) und das IOCTL-Statement (für welches wir DosDevIOctl nehmen können). Es existieren auch keine OS/2 VIO- und KBD-Headerdateien, obwohl wir sie dynamisch linken können.

Glücklicherweise ist die breite Auswahl an Grafikfunktionen von Basic unter dem geschützten Modus von OS/2 implementiert, was wiederum eine Möglichkeit darstellt, in OS/2 Grafik zu programmieren. Die schlechte Nachricht ist, daß nur der CGA-Modus unterstützt wird, und daß der Bildschirm nicht aktualisiert wird, wenn das Programm im Hintergrund läuft. Sie können zumindest derzeit Mandelbrot-Grafiken nicht im Hintergrund zeichnen lassen.

Auch laufen weder das in OS/2 enthaltene GWBasic, noch QuickBasic im geschützten Modus von OS/2.

Zu guter Letzt ist die Basic-Laufzeitbibliothek nicht reentrant, was bedeutet, daß Sie keine Multithread-Applikationen in Basic programmieren können.

Zusammenfassung

Wir haben diese vier Sprachen ziemlich kurz abgehandelt. In der Diskussion von UR/Forth fehlte die Prozeßkontrolle, Lisp besprachen wir ohne den Aspekt der künstlichen Intelligenz, und in Basic vergaßen wir das Wort Ausbildung.

Wir erhielten eine klare Antwort auf die Frage, wieso es so viele Sprachen gibt. Jede der besprochenen Sprachen beruht auf einigen guten Ideen.

Forth bietet eine vom Anwender programmierbare Syntax, die Kontrolle über den Interpreter und den Compiler Worte, die über den Stack Informationen austauschen, ja sogar alles ist ein Wort.

Lisp-Programme ähneln Listen und die Sprache manipuliert Symbole statt Variablen. Das dynamische Linken wird bis zur Laufzeit verschoben.

Modula-2 erhebt das Modul zu einem neuen Datentyp, parallele Prozesse sind Bestandteile der Sprache, und es gibt eine Synchronisation zwischen den Modulen in der Hochsprache.

Basic verwaltet den String-Speicher und die Fehlerbehandlung für Sie. Die gebräuchlichsten Operationen haben optionale Parameter und die Variablen werden automatisch deklariert. *Tabelle 2* bietet einen Vergleich der besprochenen Sprachen.

OS/2 bietet auch mit kommandozeilenorientierten Compilern eine interaktive Softwareerstellung. Sie lassen Ihren Editor in seiner eigenen Sitzung laufen und schalten zwischen Editor, Debugger und abstürzenden Programmen hin und her. Da die abstürzenden Programme weniger tödlich sind, als im Real Mode, können Sie mehr schnelle Experimente wagen.

Die Multitasking-Fähigkeiten von OS/2 und die Erweiterbarkeit mit DLLs bedeuten eine neue Generation von Compilern, Interpretern und integrierten Entwicklungsumgebungen. Dies ist schon deutlich zu sehen. Statt an einer einsamen Insel Schiffbruch erlitten zu haben, kehren wir von der Insel in die Zivilisation zurück.

Andrew Schulman

	UR/FORTH	OS2XLISP	Stony Brook Modula 2	Microsoft BASIC 6.0	C
interaktiv	x	x			
Multitasking	x		x		x
Grafik	x			x	
Schutzbehandlung		x		x	
DLL erstellen			x		x
Fehlerprüfung zur Laufzeit			x	x	
Debugging	selbst	eingeb.	M2DEBUG	CVP	CVP
mit C linken		x	x	x	

Tabelle 2: *Ein Vergleich der Sprachen.*

Anbieterinformationen

UR/FORTH Version 1.1:
Laboratory Microsystems, Inc.
P.O. Box 10430
Marina del Rey, CA 90295
(213) 306-7412

OS2XLISP (XLISP Version 2.0, OS/2-Erweiterungen 1.10):
In den USA erhältlich im Microsoft Systems Forum von Compuserve (GO MSSYS) oder über BIX.

Stony Brook Modula-2 Development System Version 1.12:
Stony Brook Software, Inc.
Forest Road, P.O. Box 219
Wilton, NH 03086
(603) 654-2525

Microsoft BASIC Compiler 6.0:
Microsoft Händler

Die Entwicklung von DOS- und OS/2-Applikationen:

Anwendungsentwicklung unter OS/2

Sie haben vor, zum Entwickeln von Anwendungen auf ein OS/2-System umzustellen, aber Sie wissen nicht, womit Sie am besten beginnen sollen. Sie haben sich ein OS/2-System gekauft, sind sich aber nicht sicher, wie Sie es am besten für die Programmierung installieren sollen. Die Multitasking-/Multithread-Fähigkeiten verwirren Sie ein wenig.

OS/2 bietet erhebliche Fortschritte in der Produktivität, egal ob Sie Programme für OS/2 oder für DOS entwickeln, aber es erfordert auch, daß Sie einige Programmierannahmen überdenken, die Sie unter DOS gemacht haben.

Dieser Artikel hilft Ihnen beim Einstieg. Zuerst sehen wir uns die Programmierumgebung von OS/2 an. Die Hardware, die Software, die Installation und die Konfiguration von OS/2 und die Nutzung von mehreren Sitzungen für die Programmentwicklung. In weiteren Artikeln in den nächsten Ausgaben werden Sie das OS/2 Application Program Interface (API) kennenlernen, das aus den Teilen für die Bildschirm-Ein-/Ausgabe (VIO), für die Tastatur (KBD) und die Maus (MOU) besteht, und werden Programme schreiben, die auf dem API aufsetzen. Wir werden das Programmieren von kombinierten Anwendungen für den Real-Adress- und den Protected-Modus kennenlernen. Am Ende fassen wir diese Ideen dann zu einem nützlichen OS/2-Programm zusammen.

Diese Reihe setzt Kenntnisse in C und DOS, aber nicht in OS/2 voraus. Unser Ziel ist die Entwicklung einiger Programme und Hilfsmittel, sowie Kenntnisse des API von OS/2 und seiner kritischen Stellen zu erlangen. Sie brauchen einen C-Compiler, das Microsoft OS/2 Programmer's Toolkit und ein OS/2-System. Am Ende können Sie sich dann der nächsten Programmierebene zuwenden: der Programmentwicklung unter dem Presentation Manager.

Welche Vorteile werden geboten?

Es ist hinreichend bekannt, daß OS/2 im Protected-Modus des 80286-Mikroprozessors arbeitet (auch der 80386 unterstützt den 80286-Protected-Modus), und Sie natürlich OS/2-Protected-Modus-Programme entwickeln können. Aber es bringt auch Vorteile, wenn Sie OS/2 als Entwicklungsumgebung für Anwendungen verwenden, die im 8088/8086-Real-Adress-Modus laufen.

Wahrscheinlich besteht der größte Vorteil von OS/2 darin, daß Sie mehrere Programmentwicklungssitzungen verwenden können. Jedes Programm kann unter OS/2 als eigenständiger Prozeß laufen, der seine eigene Bildschirmgruppe und ein logisches Tastatur- und Bildschirminterface besitzt. OS/2 verbindet dieses mit der wirklichen Tastatur und dem wirklichen Bildschirm, wenn der Prozeß im Vordergrund läuft. Prozesse, die im Hintergrund laufen, werden weiter ausgeführt, bis sie auf eine Tastatureingabe warten, die sie dann erhalten, wenn sie wieder in den Vordergrund geholt werden.

Die meisten Arbeiten beim Schreiben und Entwickeln eines Programms, einschließlich der Arbeiten die spezielle Maschinen-Ressourcen benötigen, können in mehreren Programmiersitzungen effizienter ausgeführt werden. Das Übersetzen eines Programms oder das Suchen von Textstellen mit *Grep* kann zum Beispiel im Hintergrund ausgeführt werden. Das gestattet Ihnen das Editieren eines anderen Teils des Programms oder das Übersetzen eines ganz anderen Programms.

Mit der Möglichkeit für mehrere Sitzungen in OS/2 können Sie jedem Arbeitsgang bei der Programmentwicklung eine eigene Sitzung zuordnen. Es ist möglich, verschiedene Sitzungen für den Editor, den Compiler, den Assembler, für Dienstprogramme, den Debugger und die Testumgebung zu erzeugen. Oder Sie können eine Sitzung installieren, deren einziger Zweck es ist, im Bedarfsfall Aufgaben wie das Kopieren von Disketten zu erledigen (auch das Formatieren im Hintergrund ist möglich). Sie müssen Ihren Texteditor nicht verlassen, um den Compiler zu starten. Auch brauchen Sie den Debugger nicht zu verlassen, um eine Textstelle in einer anderen Quelldatei anzusehen.

Da jede OS/2-Sitzung ihre eigene unabhängige Umgebung, ihre eigenen Pfade und ihre eigene Bildschirmkonfiguration besitzt, können Sie diese unterschiedlich, Ihren Bedürfnissen entsprechend, konfigurieren. Sie können den Pfad und die Umgebungsvariablen in der Sitzung des Compilers so einrichten, daß der Compiler die benötigten Dateien möglichst schnell findet. Sie können eine Editor-Sitzung erzeugen, wobei nur dem Editor (über Umgebungsvariablen) bestimmte Dateien und Verzeichnisse bekannt sind. Das Erzeugen von unterschiedlichen Sitzungen für die verschiedenen Aufgaben unter OS/2 ist einfach. Indem Sie die schon unter MS-DOS verfügbaren Hilfsmittel (ein paar einfache Dienstprogramme und Umgebungsvariablen) verwenden, können Sie das Bereitschaftszeichen, die Bildschirmfarbe und den Bildschirmmodus jeder Sitzung unabhängig voneinander einstellen. So können Sie sofort jede Sitzung identifizieren, sogar wenn Sie schnell zwischen ihnen hin- und herschalten.

Zum Kompilieren kann der Bildschirm zum Beispiel in den 43-Zeilenmodus mit schwarzem Hintergrund und intensivem Bereitschaftszeichen geschaltet werden. Dies gestattet einen schnellen Überblick über die Meldungen des Compilers wenn Sie umschalten. Außerdem gestattet es einen besseren Überblick, wenn Sie mehrere Quellcodedateien verwenden. Natürlich kann das Bereitschaftszeichen dieser Sitzung auch intensiv schwarz auf weißen Hintergrund sein.

Unter OS/2 können Sie jedem Programm, das eine andere Umgebung benötigt, eine eigene Sitzung zuordnen. Es ist zum Beispiel nicht ungewöhnlich, zwei Compiler gleichzeitig laufenzulassen.

Das gleichzeitige Ausführen von mehreren Editor-Sitzungen kann darüber hinaus Ihre Produktivität erhöhen.

Viele derzeit erhältliche Editoren besitzen die Fähigkeit, mehrere Dateien gleichzeitig in verschiedenen Fenstern zu bearbeiten. Indem Sie mehrere Editor-Sitzungen erzeugen, können Sie die Anzahl der bearbeiteten Dateien verdoppeln oder verdreifachen, ohne daß der Bildschirm unübersichtlich wird.

Aber gehen wir noch einen Schritt weiter: Ich habe es als sehr hilfreich empfunden, zwei gänzlich unterschiedliche Programme zu editieren, zu übersetzen und zu debuggen. Obwohl es so aussieht, als würde die Produktivität sinken, ist es doch hilfreich, wenn eine Datei besonders groß ist (über 100 Kbyte), und diese viel Zeit zum Übersetzen benötigt. So können Sie während der Übersetzung dieses Programms im Hintergrund ein anderes Programm in Ruhe im Vordergrund bearbeiten. Obwohl die Prozesse im Vordergrund eine höhere Priorität erhalten, werden Sie überrascht sein, wie schnell ein »sauberes« OS/2-Programm im Hintergrund abläuft. (Die Definition eines sauberen OS/2-Programms wird im Verlaufe der nächsten Artikel klar werden.)

Die DOS-Sitzung

OS/2 kann so konfiguriert werden, daß eine spezielle Real-Adress-Modus-Sitzung möglich ist. Diese unterstützt eine DOS 3.x-Version, die auch unter dem Namen Kompatibilitätsbox oder DOS-Sitzung bekannt ist. Diese Sitzung ist nur als Vordergrundprozeß aktiv (also bei der Anzeige auf dem Bildschirm). Sie wird im Hintergrund, also beim Umschalten zu einer OS/2-Anwendung, gestoppt, da OS/2 den Prozessor in den Protected-Modus umschaltet.

Die Programme, die in der DOS-Sitzung laufen, glauben unter DOS zu laufen, obwohl OS/2 kontrollierend eingreift. Da der Prozessor die DOS-Sitzung im Real-Adress-Modus ausführt, ist die »schnelle und unsaubere« DOS-Umgebung weiterhin vorhanden. Dies gestattet es dem DOS-Programm sich in der DOS-Sitzung »unsauber« zu benehmen und Probleme zu verursachen, was sogar schlimmere Auswirkungen hat als unter DOS. Des weiteren arbeiten einige Dienstprogramme unter DOS zur vollsten Zufriedenheit, aber nicht unter OS/2. Eine große Anzahl von Editoren zum Beispiel fragt ständig die Tastatur ab, um die Eingaben schnellstmöglich zu bearbeiten. Dies ist in einer Singletasking-Umgebung, wie DOS sie darstellt, in Ordnung. Unter OS/2 kann dadurch das System in die Knie gehen. Durch die ständige Abfrage der Tastatur von der DOS-Sitzung gibt die CPU diesem Prozeß zu viele Zeitscheiben. So bleiben diese Programme schnell, aber die Hintergrundprogramme werden um so langsamer.

Ein gut programmierter OS/2-Editor erzeugt einen Thread um die Tastatureingaben zu erledigen. Da OS/2 weiß, zu welcher Bildschirmgruppe dieser Thread gehört, blockiert es diesen Thread, bis eine Tastatureingabe für ihn ansteht. Deshalb werden die anderen Prozesse weiterhin normal ausgeführt (sowohl im Hinter- als auch im Vordergrund), und erhalten angemessene CPU-Zeit.

Obwohl die DOS-Sitzung den Umstieg auf OS/2 erleichtert, ist die Produktivität doch erheblich höher, wenn Sie OS/2-Hilfsmittel besitzen. Haben Sie zum Beispiel nur ein UNIX-ähnliches GREP-Programm für DOS, können Sie dies nur in der DOS-Sitzung laufenlassen. Sobald die DOS-Sitzung in den Hintergrund geschaltet wird, wird die Programmausführung unterbrochen.

Wäre das Programm für OS/2 programmiert, so würde es nicht nur im Hintergrund weiterlaufen, sondern es könnte auch die Vorteile von Threads (Multitasking innerhalb eines Prozesses selbst) nutzen. Ein Thread könnte zum Beispiel die Datei öffnen und suchen, ein anderer die Ergebnisse ausgeben. Dies kooperiert nicht nur mit anderen OS/2-Prozessen und bewirbt sich um die CPU und die Ressourcen, sondern ist auch entscheidend effizienter, als die DOS-Version.

Die DOS-Sitzung kann eine volle 640 Kbyte große Umgebung beherbergen. Die Einschränkungen beim Debuggen und Starten eines großen Programms sind aber dieselben wie unter DOS 3.x. Es kann unmöglich sein, ein Programm in der DOS-Sitzung mit dem Microsoft CodeView-Debugger zu testen, wenn das Programm zu groß ist, oder wenn zu viele Module mit symbolischen Informationen übersetzt sind. Dies ist aber nicht der Fall, wenn Sie ein großes Programm mit der Protected Modus-Version von CodeView debuggen, da OS/2 eine virtuelle Speicherverwaltung besitzt.

Anfangsbedingungen

Nachdem Sie beschlossen haben, Programme unter OS/2 zu entwickeln, stellt sich die Frage, was Sie benötigen, um zu beginnen. Und was fällt in die Kategorie, »Es wäre großartig, wenn ich es mir leisten könnte«?

Unser Ziel ist es, möglichst professionell beim Programmieren unter OS/2 zu werden. Es gibt dorthin aber zwei Wege, einen ausreichenden und einen bevorzugten. Es drängt sich die Frage nach dem Unterschied zwischen erster und zweiter Klasse auf. Der ausreichende Weg enthält alles, um erfolgreich OS/2-Programme zu erstellen. Die erste Klasse ist natürlich teurer, macht aber den Unterschied aus zwischen »installieren und beginnen« und »installieren und sehr schnell beginnen«. Für beide Wege benötigen Sie einen Computer mit wenigstens einer Intel 80286- oder 80386-CPU, 2,5 Mbyte Arbeitsspeicher, 20 Mbyte Festplattenkapazität, ein Diskettenlaufwerk mit hoher Schreibdichte (entweder 1,44 Mbyte, 3,5 Zoll oder 1,2 Mbyte, 5,25 Zoll) und natürlich einen Monitor und eine Tastatur.

Die folgenden Teile sind nützlich (und wertvoll): 3 oder mehr Mbyte Arbeitsspeicher, 40 oder mehr Mbyte Festplattenkapazität, eine IBM EGA- oder VGA-Karte oder 100% kompatible Grafikkarte (die EGA-Karte sollte mindestens 128 Kbyte RAM besitzen) und eine Maus.

OS/2-Befehlsübersicht

Die meisten MS-DOS-Kommandos sind im Protected-Modus von OS/2 weiterhin vorhanden. Jedoch wurden viele verbessert und einige neue hinzugefügt.

Die folgenden Kommandos wurden gegenüber den DOS- und den Real-Modus-Versionen verbessert. Sie erlauben jetzt mehrere Argumente.

DEL	MD/MKDIR
DIR	RD/RMDIR
TYPE	VOL

Sie können zum Beispiel alle .C-, .H- und .ASM-Dateien in einem Verzeichnis mit dem folgenden Kommando löschen:

```
DEL *.C *.H *.ASM
```

(Ich rate Ihnen aber nicht unbedingt diesen Befehl zu testen.)

Das MODE-Kommando erlaubt jetzt die Angabe eines Timeout-Wertes und die Art des Hardwarehandshakes.

Die folgenden Kommandos sind neu, und im Protected-Modus von OS/2 verfügbar:

&&

&& dient zum bedingten Aneinanderreihen von Kommandos.

```
DIR X && DEL X
```

Dieses Kommando löscht X nur, wenn der erste Befehl erfolgreich war.

&

& dient zum unbedingten Aneinanderreihen von Kommandos.

```
DIR X & DEL X
```

Hier versucht OS/2 beide Kommandos auszuführen. Das Ergebnis eines Kommandos beeinflußt nicht die Ausführung des anderen.

||

|| kann bedingt Kommandos ausführen.

```
COPY X Y || COPY A Y
```

Das zweite Kommando wird nur ausgeführt, wenn das erste Kommando nicht erfolgreich war.

()

Klammern können Gruppen von Kommandos zusammenfassen.

```
DEL X || (ATTRIB -R X & DEL X)
```

Ist das Löschen der Datei nicht erfolgreich (weil das Lese-/Schreib-Attribut von X gesetzt ist), versucht OS/2 das Lese-/Schreib-Attribut von X zu löschen, und falls dies erfolgreich ist, die Datei zu löschen.

^ Das Zirkumflex oder Controlzeichen ^ normalisiert Sonderzeichen.

```
^> FILE.TXT
```

wird als "> FILE.TXT" aufgefaßt. OS/2 erkennt > nicht als Sonderzeichen und leitet keine Dateiumleitung ein.

ANSI[ON][OFF]

Dieses Kommando schaltet die Unterstützung der ANSI-Steuersequenzen um. Es wird statt des Real-Modus-Treibers ANSI.SYS verwendet.

CMD [Optionen][Argumente]

Dient zum Starten eines Kommandoprozessors im Protected-Modus. Als Optionen können /c und /k angegeben werden. /c übergibt am Ende der Ausführung die Kontrolle an den primären Kommandoprozessor, /k an einen neuen.

CREATEDD

Erzeugt ein Speicherabbild der Diskette.

DETACH [Programm[Argumente]]

Dieses Kommando führt Protected-Modus-Programme und -Prozesse im Hintergrund aus. Programme im Hintergrund können aber keine Eingaben von der Tastatur erhalten, und alle Bildschirmausgaben gehen verloren. Ist das Kommando erfolgreich, so wird die Prozeß-ID zurückgegeben.

DPATH path;...

Ist ähnlich dem APPEND-Kommando im Real-Adress-Modus. DPATH erzeugt einen Pfad, den Applikationen zum Eröffnen von Dateien über die Umgebungsvariable DPATH suchen können. Applikationen die DPATH benutzen, müssen die Umgebung nach dem Wert durchsuchen.

ENDLOCAL

Wird bei der Batchdateiverarbeitung benötigt. Dieses Kommando setzt wieder die bei einem vorangegangenen SETLOCAL-Kommando gespeicherten Werte für das Laufwerk und das Unterverzeichnis.

EXTPROC

Dieses Kommando spezifiziert einen anderen Kommandoprozessor für die Bearbeitung von Batchdateien.

KEYB

Gibt eine Tastenanordnung für eine Tastatur an, die nicht der US-Norm entspricht.

SETLOCAL

Wird bei der Batchdateiverarbeitung benötigt. Dieses Kommando merkt sich das aktuelle Laufwerk und das aktuelle Unterverzeichnis für ein späteres Zurückspeichern mit ENDLOCAL.

SPOOL

Aktiviert den Hintergrund-Spooler.

START "Sitzungsname" [/c] [Kommando [Optionen]]

Erzeugt eine neue Sitzung im Protected-Modus. Der Name der Sitzung (angegeben als Argument) erscheint im Menü des Programm-Selektors.

TRACE <ON|OFF>[CODE[,...]]

Schaltet das Tracen des Systems ein bzw. aus.

TRACEFMT

Zeigt den Inhalt des Trace-Puffers des Systems in LIFO-Reihenfolge an.

Im Falle der Software haben Sie eine größere Auswahlmöglichkeit. Um Programme unter OS/2 zu entwickeln, benötigen Sie die folgenden Hilfsmittel: einen Compiler (C oder Pascal, etc.) der OS/2 unterstützt, einen Assembler, einen Linker für den Protected-Modus, einen Editor für den Protected-Modus, einen Debugger, ebenso für den Protected-Modus, und andere Hilfsmittel, wie MAKE, GREP etc. und die OS/2-Version 1.0 (die Version 1.1, die den Presentation Manager enthält, funktioniert auch, ist aber am Anfang nicht notwendig).

Wie Sie sicherlich wissen, bietet Microsoft die meisten Compiler schon für den Protected-Modus an. So zum Beispiel den optimierenden C-Compiler in der Version 5.1. Dies ist eine »BOUND-Applikation«, die Programme erzeugen kann, die unter beiden Betriebssystemen laufen, das heißt, entweder im Real-Adress-Modus oder im Protected-Modus. Wenn Sie ein erfahrener C-Programmierer sind, so werden Sie einige Assemblerfunktionen besitzen. Der Microsoft Makro-Assembler der Version 5.1 (MASM) erfüllt die Anforderungen (wie der C-Compiler ist es ein kombiniertes Programm). Der neue inkrementelle Linker LINK4 von Microsoft linkt Objektdateien (inkrementell und daher erheblich schneller) zu einem ausführbaren Programm sowohl für DOS als auch für OS/2. Er ist in Microsoft C 5.1 und auch im MASM 5.1 enthalten.

Auch der Microsoft-Editor, ein mächtiges Protected-Modus-Programm ist sowohl in Microsoft C, als auch im MASM enthalten. Ich möchte auch noch darauf hinweisen, daß so ausgezeichnete Produkte wie Lugarus Epsilon oder der ME-Editor von Magma Systems erhältlich sind, oder in Kürze erscheinen.

Toolkit oder SDK?

Microsoft bietet zwei Wege, um die benötigten Hilfsmittel zum Entwickeln von OS/2-Programmen zu bekommen. Das Microsoft OS/2 Software Development Kit, kurz SDK genannt, bietet eine umfangreiche Sammlung von allen benötigten Hilfsmitteln. So sind vom OS/2 (Versionen 1.0 und 1.1, die letztere enthält den LAN-Manager und den Presentation Manager) bis zum C-Compiler und Assembler alle Programme enthalten. Dies ist das bevorzugte Paket, aber die Kosten sind für viele Entwickler zu hoch.

Die Alternative besteht im Erwerb der einzelnen Teile, die Sie benötigen. Leser des *Microsoft System Journals*, die schon die neuesten Versionen von Microsoft C und/oder MASM besitzen, haben es leichter, da die meisten der benötigten Hilfsmittel mit den Produkten geliefert werden. Diese und OS/2 selbst sind zum Entwickeln von OS/2 Programmen ausreichend. Sie sollten sich aber auch das Microsoft OS/2 Programmer's Toolkit (PTK) zulegen. Das PTK, das auch im SDK enthalten ist, enthält viele der wichtigen Hilfsmittel und Beispielprogramme des SDK. Am wichtigsten ist aber der komplette Programmer's Reference Guide für alle OS/2-API-Funktionsaufrufe.

Wenn Sie noch nicht beabsichtigen, Presentation Manager- oder LAN Manager-Anwendungen zu entwickeln (was am Anfang sehr weise ist), so werden Sie sich möglicherweise für das erheblich günstigere PTK entscheiden. Bedenken Sie aber, daß Sie hierbei noch einen Hochsprachen-Compiler (wie etwa C 5.1) einen Editor und OS/2 selbst benötigen.

Die Installation von OS/2

Wenn Sie dazu bereit sind, OS/2 zu installieren, so bieten sich Ihnen eine Reihe von Optionen, die primär davon abhängen, ob Sie schon MS-DOS auf der Maschine installiert haben, auf der Sie OS/2 installieren wollen. Diese Optionen sind:

- Standardmäßig OS/2 starten (MS-DOS muß von der Diskette gestartet werden)
- Standardmäßig MS-DOS starten (OS/2 muß von der Diskette gestartet werden)
- Abfrage, mit welchem Betriebssystem Sie starten wollen (Dual-Boot-Version, die in früheren SDK-Versionen enthalten war, und auch heute noch von einigen Vertreibern, wie etwa Zenith, angeboten wird.)

Wenn Sie ab und zu DOS noch starten wollen, so dürfte die erste Version für Sie die richtige sein. Die letzte Möglichkeit dürfte die bequemste sein. Bedenken Sie auch, daß bei einem laufenden OS/2-System die DOS-Sitzung ja noch verfügbar ist. Die Auswahl beschränkt sich oft auf den Start von OS/2 oder MS-DOS 3.x von der Festplatte. Nicht alle OS/2-Versionen bieten die Möglichkeit des Dual Boot bei der Installation. Diese Möglichkeit ist auch in der von IBM vertriebenen Version von OS/2 nicht enthalten.

Der Einfachheit halber schlage ich vor, daß Sie OS/2 auf Ihrer MS-DOS-Entwicklungsmaschine installieren, und auch in der Lage sind MS-DOS ganz ohne OS/2 laufen zu lassen – mit anderen Worten die Dual Boot-Installation. Es spielt in Wirklichkeit keine Rolle, welche Installation Sie auswählen, die bequemste sollte ausgewählt werden. Obwohl es noch andere Schritte gibt, die von der Größe und der Anzahl der Festplatten in Ihrem System abhängen, besteht die OS/2-Installation nur aus dem Starten des Computers mit der OS/2-Installationsdiskette und dem Beantworten einiger Fragen (es ist einfacher als die Installation von C 5.1 mit seiner großen Anzahl an Möglichkeiten). Das Installationsprogramm fordert Sie bei Bedarf auf, andere Disketten einzulegen. Das ist schon alles.

Wenn Sie die Dual-Boot-Option installieren, dann verändert das Installationsprogramm den Bootsektor Ihrer Festplatte. Nach dem Starten des Computers erscheint jetzt die folgende Meldung auf dem Bildschirm:

```
Boot: Enter=OS/2, ESC=DOS
```

Das ist alles, was für die Wahl zwischen DOS und OS/2 zu tun ist. Wenn Sie sich für die Installation von OS/2 entscheiden, dann startet OS/2 direkt.

Die Konfiguration von OS/2

Ebenso wie mit CONFIG.SYS unter MS-DOS kann der Anwender auch OS/2 konfigurieren, indem er Anweisungen in die Datei CONFIG.OS2 schreibt. Einige wie BREAK und FCBS sind identisch mit denjenigen unter DOS und sind im Real-Adress-Modus von OS/2 immer noch verfügbar. Andere, wie BUFFERS, DEVICE und SHELL, sind auch für den Protected-Modus zuständig. DEVICE wird für die Installation der OS/2-Einheitentreiber benötigt. In CONFIG.OS2 bezeichnet der Eintrag SHELL den Real-Modus-Kommandoprozessor, ebenso wie dies unter MS-DOS der Fall ist.

Zusätzlich sind aber unter OS/2 eine Reihe weiterer Kommandos verfügbar:

DISKCACHE installiert Cache-Speicher für den Massenspeicherzugriff und legt dessen Größe fest. Standardmäßig ist der Cache-Speicher nicht aktiviert.

IOPL gibt den Protected-Modus-Anwendungen, die sie anfordern, mehr Ein-/Ausgabe-/Datenprivilegien. IOPL=YES muß in der Datei stehen, um CVP, die Protected-Modus-Version von CodeView, laufenzulassen. Die Voreinstellung ist YES.

LIBPATH ist ein CONFIG.OS2-Kommando ähnlich den Kommandos PATH und DPATH. Es gibt das Unterverzeichnis für die dynamischen Linkbibliotheken an. Das voreingestellte Verzeichnis ist das Hauptverzeichnis des Bootlaufwerks.

MAXWAIT gibt die maximale Zeit in Sekunden an, die gewartet wird, bevor der Scheduler die Priorität eines nicht mehr bedienten Prozesses erhöht. Voreingestellt ist 3. Dieser Wert kann zwischen 1 und 255 variiert werden.

MEMMAN aktiviert oder deaktiviert die virtuelle Speicherverwaltung. Das Aktivieren von SWAP gestattet OS/2, mehr Prozesse laufen zu lassen, als in den Speicher passen. Dies setzt auch automatisch MOVE. Der Parameter MOVE gestattet OS/2, Segmente im Speicher zu verschieben. Die Leistungsfähigkeit steigt bis zu einem gewissen Maße, wenn die virtuelle Speicherverwaltung deaktiviert ist. Voreingestellt sind SWAP,MOVE wenn das Bootlaufwerk eine Festplatte ist, und NOSWAP/NOMOVE, wenn es ein Diskettenlaufwerk ist.

PAUSEONERROR läßt OS/2 die Fehlermeldungen anzeigen, während es die Datei CONFIG.OS2 bearbeitet. Voreingestellt ist YES.

PRIORITY beeinflußt die Art und Weise, in der ein Prozeß seine Priorität verändert. OS/2 unterscheidet drei Prioritätsklassen: zeitkritisch, normal, und träge. In jeder Klasse gibt es 32 Stufen. Die Prioritäten in den normalen Klassen können angepaßt werden.

Das Setzen von PRIORITY=ABSOLUTE in der Datei CONFIG.OS2 verhindert das Ändern der Priorität in der Klasse normal.

Ist PRIORITY=DYNAMIC eingestellt, so versucht OS/2 bei jeder Zeitscheibe demjenigen Prozeß die höchste Priorität zu geben, der die meisten CPU-Ressourcen benötigt. Hat ein Prozeß eine niedrigere Priorität als er haben sollte, so setzt OS/2 diese höher. Voreingestellt ist PRIORITY=DYNAMIC

PROTECTONLY läßt Anwendungen sowohl im Protected-Modus, als auch im Real-Adress-Modus laufen, wenn PROTECTONLY=NO eingestellt wurde. Ist PROTECTONLY=YES, so können Sie keine Real-Adress-Modus-Anwendungen laufenlassen, es steht Ihnen aber dann der von der Kompatibilitätsbox benötigte Speicher (bis zu 640 Kbyte) für Protected-Modus-Anwendungen zur Verfügung. Voreingestellt ist NO.

PROTSHELL gibt den Protected-Modus-Kommandoprozessor und den Programm-Selektor an. Voreingestellt ist:

```
PROTSHELL=SHELL.EXE CMD.EXE /K OS2INIT.CMD
```

REM erlaubt Ihnen das Einfügen von Kommentaren in die Datei CONFIG.OS2

RMSIZE gibt die Größe des Speichers an, der für die Kompatibilitätsbox zur Verfügung steht. Die Voreinstellung ist von der Gesamtgröße abhängig. In der Regel sind dies 512 oder 640 Kbyte.

RUN gestattet es während des Systemstarts Programme im Protected-Modus zu starten. So kann mit einigen RUN=-Kommandos in der Datei CONFIG.OS2 eine Anzahl von Programmen gestartet werden, bevor der Programm-Selektor aktiv wird. Beachten Sie, daß OS/2 alle Einheitentreiber lädt, bevor ein Programm gestartet wird, und daß Batchdateien (.CMD) nicht mit RUN gestartet werden können.

SWAPPATH gibt den Ort der Swapdatei an (es sollte eine Festplatte sein), der Datei also, die OS/2 für die Implementierung der virtuellen Speicherverwaltung benötigt. Die kleinste Größe ist 640 Kbyte. Ist die virtuelle Speicherverwaltung deaktiviert (MEMMAN), so wird dieses Kommando ignoriert.

THREADS gibt die maximale Anzahl der Threads zu einem Zeitpunkt an. Ein Thread ist ein Teil einer Anwendung oder eines Prozesses, der unter OS/2 eigenständig laufen kann. Ein Prozeß besteht immer aus mindestens einem Thread, kann aber mehrere enthalten. Dies entspricht der Erfüllung von Teilaufgaben in jedem Prozeß.

Bedenken Sie, daß OS/2 ungefähr 14 Threads benötigt. Das System setzt die Voreinstellung, und jeder zusätzliche Thread benötigt zusätzlich etwas Speicherbereich.

TIMESLICE ist das Zeitintervall, das OS/2 einem Thread eines Prozesses zusammenhängend zur Verfügung stellt. Dieses Kommando gibt die minimale und die maximale Zeit in Millisekunden an, die OS/2 einem Prozeß zur Verfügung stellt, bevor es andere Prozesse bedient. Ist nur ein Wert angegeben, so sind beide Werte gleich.

Der minimale Wert muß größer als 31 sein. Der maximale Wert muß gleich oder größer dem minimalen sein. Das Setzen von TIMESLICE=500 veranlaßt den Scheduler, eine halbe Sekunde nach dem Starten eines Threads zu warten, bis er nachsieht, ob ein anderer Thread eine höhere Priorität besitzt.

TRACE OS/2 merkt sich die Aktionen, die es bei der Bearbeitung von Interrupts und Funktionen durchführt, was bei der Programmentwicklung hilfreich sein kann. Das Tracen des Systems kann durch die Kommandos TRACE=ON oder TRACE=OFF in der Datei CONFIG.OS2 an- oder abgeschaltet werden. Sie können spezielle Arten von Ereignissen tracen, wenn Sie nach dem ON den Code angeben. Sie können TRACE=ON gefolgt von TRACE=OFF X,Y angeben, wobei alle Ereignisse, außer diejenigen mit den Codes X und Y, aufgezeichnet werden. Voreingestellt ist TRACE=OFF. Der Ereigniscode muß zwischen 0 und 255 liegen.

TRACEBUF gibt die Größe des Tracebuffers in Kbyte an.

Zusätzlich zu diesen Kommandos sind auch noch folgende für den internationalen Einsatz verfügbar:

CODEPAGE wählt die Codepage.

COUNTRY wählt das Format von Datum, Uhrzeit und das Währungszeichen.

DEVINFO bereitet eine Einheit für die Verwendung der Codepage vor.

Beachten Sie, daß OS/2 die Statements FILES und LASTDRIVE, die in der Datei CONFIG.SYS in DOS üblich sind, einfach ignoriert.

Beispiele für CONFIG.OS2-Dateien

Es folgen zwei Beispiele von CONFIG.OS2-Dateien. Die erste gilt für die Version 1.0 von OS/2, die zweite gilt für die Version 1.1, die den Presentation Manager enthält.

Beispiel 1 (OS/2 Version 1.0)

```
REM CONFIG.OS2 FUER OS/2 V.1.0
BUFFERS=50
SHELL=C:\COMMANDO.COM /P /E:500
PROTSHELL=C:\SHELL.EXE CMD.EXE /K C:\OS2INIT.CMD
LIBPATH=C:\OS2\DLL
DEVICE=C:\OS2\DEV\MOUSEA03.SYS
DEVICE=C:\OS2\DEV\POINTDD.SYS
DEVICE=C:\OS2\DEV\COM01.SYS
RUN=C:\OS2\PBIN\SPOOL.EXE C:\OS2\SPOOL /O:LPT1
BREAK=ON
FILES=25
IOPL=YES
TRACE=ON
TRACEBUF=8
RUN=C:\CMD.EXE /C C:\CSET
DISKCACHE=512
```

Beispiel 2 (OS/2 Version 1.1)

```
REM CONFIG.OS2 FUER OS/2 V.1.1 UND PM
BUFFERS=30
SHELL=C:\OS2\RBIN\COMMANDO.COM /P /E:1024 C:\OS2\RBIN
REM Der folgende Eintrag muß in einer Zeile stehen
PROTSHELL=C:\OS2\PMSHELL.EXE C:\OS2\PBIN\CMD.EXE /K
C:\OS2INIT.CMD
RMSIZE=640
PROTECTONLY=NO
BREAK=OFF
FCBS=16,8
THREADS=64
IOPL=YES
LIBPATH=C:\OS2\DLL
SET PATH=C:\OS2\;C:\OS2\BIN;C:\OS2\PBIN
MEMMAN=SWAP,MOVE
DISKCACHE=64
MAXWAIT=3
SWAPPATH=C:\
DEVICE=C:\OS2\DEV\MOUSEA03.SYS
DEVICE=C:\OS2\DEV\POINTDD.SYS
DEVICE=C:\OS2\DEV\PMDD.SYS
DEVICE=C:\OS2\DEV\COM01.SYS
DEVICE=C:\OS2\DEV\EGA.SYS
RUN=C:\OS2\PBIN\SPOOL.EXE C:\OS2\SPOOL /O:LPT1
BREAK=ON
FILES=25
IOPL=YES
TRACE=ON
TRACEBUF=8
RUN=C:\CMD.EXE /C C:\CSET
DISKCACHE=512
```

Die OS/2-Dateien

Wenn Sie sich Ihr System nach der Installation von OS/2 ansehen, dann bemerken Sie eine Reihe von Veränderungen. Sie sehen einige neue Unterverzeichnisse im Hauptverzeichnis des Startlaufwerks, wie in *Abbildung 1* gezeigt. Jedes Unterverzeichnis erfüllt einen besonderen Zweck, wie angegeben. Möglicherweise ändert sich diese Verzeichnisstruktur in der Zukunft etwas, aber so wie sie gezeigt wird, wird sie im folgenden auch verwendet.

Das Installationsprogramm kopiert eine Menge Dateien in das Hauptverzeichnis des Startlaufwerks oder in die oben erwähnten Unterverzeichnisse. Obwohl die meisten dieser Dateien OS/2-Dienstprogramme sind, bedürfen einige doch einer Erklärung. Zuerst sehen Sie zwei versteckte Dateien, wie in einer DOS-Umgebung: OS2BIO.COM und OS2DOS.COM. (Die Systemdateien von DOS, IBMBIO.COM und IBMDOS.COM, sind unversehrt). Sie befinden sich immer im Hauptverzeichnis, wenn die Dual-Boot-Installation gewählt wurde. Andere OS/2-Anbieter können aber andere Dateinamen für diese Dateien gewählt haben.

Die nächsten Änderungen sind unter dem Aspekt des friedlichen Nebeneinander von DOS und OS/2 wichtig.

CONFIG.OS2 gleicht der Datei CONFIG.SYS unter MS-DOS. OS/2 bearbeitet sie beim Start des Systems. Wenn Sie nicht im Dual-Boot-Modus arbeiten, so heißt diese Datei auch unter OS/2 CONFIG.SYS

STARTUP.CMD ist das OS/2-Äquivalent der Datei AUTOEXEC.BAT unter MS-DOS. Sie wird nach der Bearbeitung von CONFIG.SYS ausgeführt und gestattet Ihnen das Starten von anderen Programmen und Prozessen sowie das Verändern der Umgebung von OS/2.

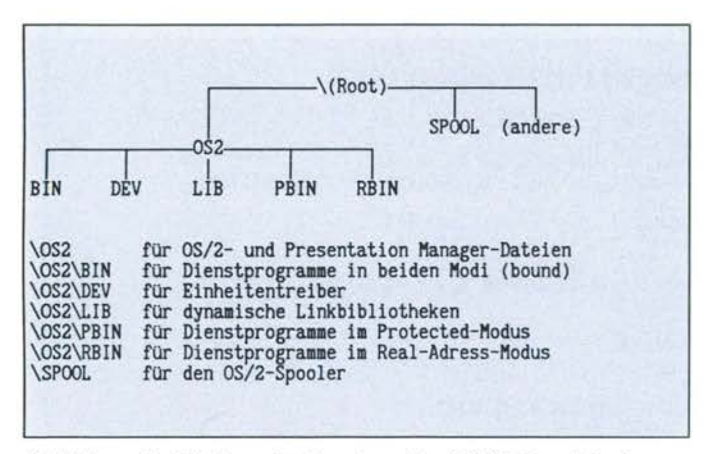


Abbildung 1: *Die Standardstruktur der OS/2-Verzeichnisse.*

OS2INIT.CMD ist die Batchdatei, die immer beim Start einer neuen Sitzung im Proteced-Modus ausgeführt wird.

AUTOEXEC.OS2 läuft wie die Datei AUTOEXEC. BAT unter MS-DOS bei der Initialisierung der Real-Adress-Modus-Sitzung und dient zur Konfiguration der Umgebung der DOS-Sitzung. Diese Datei behält unter OS/2 den Dateinamen AUTOEXEC.BAT, wenn der Dual Boot-Modus nicht installiert ist.

Zwei dieser Dateien sind OS/2-Batchdateien. OS/2 verlangt, daß Batchdateien den Dateizusatz CMD verwenden, um sie von MS-DOS Batchdateien unterscheiden zu können. Die folgenden Dateien sind unter den verbleibenden Dateien, die bei der Installation hinzugefügt wurden, und erfordern eine Erläuterung:

CMD.EXE ist der OS/2-Kommandointerpreter (wie COMMAND.COM unter MS-DOS)

HARDERR.EXE ist der Behandlungsprozessor für kritische Fehler unter OS/2.

PMSHELL.EXE ist der Task-Manager des Presentation Managers (für OS/2 Version 1.1).

SHELL.EXE ist der OS/2 Programm-Selektor (für OS/2 Version 1.0).

SWAPPER.DAT ist die Swapdatei für den virtuellen Speicherverwalter.

Die Konfiguration von OS/2

Es gibt verschiedene Möglichkeiten, OS/2 mit der Datei CONFIG.OS2 zu konfigurieren. Sie möchten sicherlich die benötigten Einheitentreiber (mit DEVICE=) laden, und sie möchten Pfade zu dem Real-Modus-Kommandoprozessor (SHELL= wie unter DOS) und zu dem Protected Modus-Kommandoprozessor setzen. Das Letztere erfolgt wie in *Abbildung 2A* dargestellt. SHELL.EXE ist der OS/2-Programm-Selektor (OS/2 Version 1.0) und CMD.EXE ist der OS/2-Kommandoprozessor. Da eines der Argumente von SHELL.EXE CMD.EXE ist, benützt der erste den zweiten zum Laden und Laufenlassen von Programmen.

2A

```
PROTSHELL=C:\SHELL.EXE C:\CMD.EXE /K C:\OS2INIT.CMD
```

2B

```
PROTSHELL=C:\OS2\PMSHELL.EXE C:\OS2\PBIN\CMD.EXE /K
C:\OS2INIT.CMD
```

2C

```
RUN=C:\OS2\PBIN\SPOOL.EXE
```

2D

```
RUN=C:\CMD.EXE /C C:\OS2\PBIN\MESTART.CMD
```

2E

```
START "HyperACCESS" /c ha
START "Paradox" /c pdoxos2 -multi
START "OS/2 Command Line"
```

Abbildung 2: *Beispieleinträge in der CONFIG.SYS-Datei von OS/2.*

Um alle Fälle abzudecken, möchte ich hinzufügen, daß die SDKs, die ab Oktober 1988 ausgeliefert werden, Installationsprogramme für eine Beta-Version des Presentation Managers enthalten, der in der Version 1.1 enthalten sein wird. Die Vorabversion des PM ist relativ funktional, und wenn Sie sein Startprogramm und seinen Taskmanager installieren (beide sind die Alternativen des PM zum Programm-Selektor), so könnte Ihr PROTSHELL-Eintrag wie im *Abbildung 2B* aussehen.

Die folgenden Vorschläge sollten Sie beherzigen, wenn Sie Programme unter OS/2 entwickeln:

- Benutzen Sie das Disk-Cache-Programm von OS/2, um die Verarbeitungsgeschwindigkeit zu erhöhen. Sie können dies erreichen, indem Sie den Eintrag DISKCACHE=x in die Datei CONFIG.OS2 einfügen. Dabei steht x für die Größe in Kbyte. Ich schlage mindestens 256 vor (bei einer Entwicklungsmaschine mit 2,5 Mbyte RAM).
- Um die Protected-Modus-Version von CodeView laufenzulassen, benötigen Sie den Eintrag IOPL=YES in der Datei CONFIG.OS2, der solchen Programmen wie CodeView zusätzliche Ein-/Ausgabe-/Datenprivilegien gibt.
- Wenn Sie dynamische Linkbibliotheken einsetzen oder erzeugen wollen, sollten Sie die Anweisung LIBPATH=*Pfad* aufnehmen, wobei *Pfad* auf das Verzeichnis zeigt, in dem die DLLs sich befinden. (Die meisten Installationsprogramme erledigen dies für Sie.)
- Wollen Sie im Falle einer nicht benötigten DOS-Sitzung Speicher sparen, dann verwenden Sie PROTECTONLY=YES in der Datei CONFIG.OS2. Dies verhindert das Anlegen einer DOS-Sitzung, auch wenn Sie die Anweisung SHELL= verwenden. Als Alternative können Sie auch eine DOS-Sitzung behalten, aber deren

Das OS/2-SDK (Software Development Kit)

Das OS/2-SDK ist die vollständigste Quelle für Entwicklungs-Hilfsmittel und -Material. Obwohl es nicht ganz billig ist, kann es doch sein, daß Sie es benötigen, je nachdem welche Art von OS/2-Anwendungen Sie entwickeln. Das SDK enthält alle benötigten Hilfsmittel in einem. Wenn Sie Presentation Manager- oder LAN Manager-Anwendungen entwickeln wollen, so enthält das SDK alle notwendigen Teile. Wollen Sie aber keine Presentation Manager- oder LAN Manager-Anwendungen entwickeln, so können Sie auch die benötigten Komponenten einzeln kaufen.

Sie sehen im Anschluß eine Liste der enthaltenen Teile. Bedenken Sie, daß zusätzlich zum Presentation Manager, LAN-Manager und dem OS/2 Programmer's Toolkit sowohl Windows, als auch das Windows-SDK enthalten sind. Das SDK enthält diese, da die Programmierschnittstelle von Windows der des Presentation Managers gleicht. Eine Regel lautet: Versuchen Sie die Programmierung unter Windows zu verstehen. Haben Sie dies erreicht, so ist die Programmierung einer Anwendung für den Presentation Manager sehr viel einfacher.

- MS OS/2 Version 1.0; MS OS/2 Version 1.1 (mit Presentation Manager); MS Windows 2.03; MS OS/2 Version 1.1 Toolkit
- Tools: Microsoft C Version 5.1; Microsoft Assembler Version 5.1; CodeView (Real Adress und Protected Modus);
  Utilities: Editor, Linker, Bibliotheksverwalter Bind (für Anwendungen in beiden Modi), Make, Grep, inkrementeller Linker.
- QuickHelp für OS/2
- OS/2 Programmer's Toolkit (PTK) Version 1.0
- MS OS/2 LAN-Manager: User's Guide; Programmer's Reference.
- MS OS/2 Presentation Manager: Conversion Guide; Reference, Volume I; Reference Volume II; Reference Addendum; »Programming the OS/2 Presentation Manager« (und Begleitdiskette) von Charles Petzold
- MS Windows Version 2.03 Software Development Kit (SDK); Learning Guide; Tools, Style Guide, Extension; Programmer's Reference Guide; Windows Applications Tools.
- Inside OS/2 von Gordon Letwin.

Das OS/2 PTK (Programmer's Toolkit)

Das OS/2 Programmer's Toolkit von Microsoft enthält viele der benötigten Hilfsmittel, um komplexe Protected-Modus-Anwendungen unter OS/2 zu erstellen. Das PTK enthält die notwendigen Hilfsmittel, um OS/2-Programme zu erstellen. Enthalten sind:

- Setup Guide und User's Guide.
- Programmer's Learning Guide und Programming Tools, die in das Programmieren in C unter OS/2 einführen. Sie setzen Erfahrungen in beidem voraus. Hier ist auch eine Abhandlung über das Erstellen von dynamischen Linkbibliotheken enthalten.
- Programmer's Reference enthält alle OS/2 Systemaufrufe.
- QuickHelp
- Beispielprogramme die folgende Themen demonstrieren: Speicherallokierung, Programmierung von mehreren Threads, Thread-Beispiele in Assembler und C, Parameterübergabe an Threads, kritische Programmteile in Threads, der Gebrauch von DosKill und DosExit, die Steuerung des Lautsprechers, der Zugriff auf die Maschinenkonfiguration und den Maschinenmodus, die Landesinformation, Zugriff auf Datum und Uhrzeit; dynamische Linkbibliotheken in C und Assembler, der Gebrauch von Exitlist; der Zugriff auf die Laufwerkinformationen; die Umgebungsinformationen; hello.c für OS/2; der Zugriff auf die Prozeßinformationen; die Tastatureingabe, der Gebrauch von Monitoren, Pipes und Queues; der Zugriff auf die Dateihandle-Informationen, der Gebrauch von Sitzungen, gemeinsamer Speicher, Signalbearbeitung; das Eintragen eines VIO-Untersystems; eine einfache Terminalemulation; eine einfache bildschirmorientierte Bearbeitung; Textsuche mit mehreren Threads; Wechseln der Verzeichnisse; Anzeige von Dateien; LIFE-Spiel; EGA-Grafik; Mausschnittstelle.
- Die Programmentwicklungs-Hilfsmittel enthalten neben den benötigten Headerdateien und Bibliotheken Dienstprogramme für die Erstellung von Anwendungen in beiden Modi; das Suchen in Bibliotheken; das Suchen von Dateien; das Erzeugen von Import-Bibliotheken; das Anzeigen von .EXE-Dateiheadern; Aktivieren des System Trace; Binden von Nachrichten; Anzeige des gemeinsamen Speichers und Dateien; Suche von Dateien; Initialisierung der Tastaturverzögerung und der Wiederholrate.

Größe begrenzen. Dies wird durch die Zeile RMSIZE=x erreicht, wobei x die Speichergröße bis 640 ist.

- Wenn Sie Programme mit vielen Threads laufenlassen, dann können Sie die zulässige Anzahl der Threads mit dem Kommando THREADS=x erhöhen, wobei x die zulässige Anzahl der Threads ist. Bedenken Sie, daß es einen geringer Speicherplatzbedarf relativ zur Gesamtzahl der Threads gibt.

RUN und START

OS/2 gestattet es Ihnen, Programme am Beginn mit den Kommandos RUN und START zu starten. RUN ist eine Anweisung, die in der Datei CONFIG.OS2 ein Programm oder einen Prozeß starten kann. Die Zeile in *Abbildung 2C* zum Beispiel startet den Druckerspooler, wenn OS/2 die Datei CONFIG.OS2 abarbeitet. Da in der Datei CONFIG.OS2 mehrere RUN-Anweisungen stehen dürfen, können einige Prozesse direkt von CONFIG.OS2 starten.

RUN unterliegt aber einer Einschränkung: Es besteht keine Möglichkeit CMD-Dateien auszuführen. So kann das Programm seine Umgebung nicht verändern. Sie können dies aber umgehen, indem Sie die CMD-Datei als Argument des OS/2-Kommandoprozessors CMD.EXE angeben. Sie können also eine Zeile wie in *Abbildung 2D* verwenden, um eine CMD-Datei auszuführen. Sie können aber auch das Kommando START benutzen. START gestattet Ihnen die Aktivierung einer weiteren Sitzung in der OS/2-Kommandozeile. Wenn Sie eine CMD-Datei erzeugen wollen, die verschiedene Prozesse startet (ohne jeden Prozeß über den Programm-Selektor anwählen zu müssen), dann kann die Datei Anweisungen wie in *Abbildung 2E* enthalten. Wollen Sie eine Reihe von Programmen starten, dann ist die bevorzugte Methode das START-Kommando in der Datei STARTUP.CMD.

Wechseln der Programme

Das Wechseln zwischen den Programmen in OS/2 ist einfach. Sie können immer [Ctrl][Esc] eingeben, um in den Programm-Selektor zu gelangen (unter dem PM den Taskmanager), der eine Liste von zu startenden Programmen und einer Liste von aktiven Sitzungen anzeigt. Sie wählen dann die Sitzung, zu der Sie wechseln wollen, oder den Prozeß, den Sie starten wollen. Ich finde es leichter, in den aktiven Sitzungen mit der Tastenkombination [Alt][Esc] zu wechseln. Dies gestattet Ihnen den schnellen Wechsel von Sitzung zu Sitzung, bis Sie diejenige erreicht haben, in der Sie arbeiten wollen. Dies funktioniert sehr schnell auch unabhängig von der Hardware, sogar wenn Sie über eine DOS-Sitzung wechseln. Es funktioniert sogar ziemlich schnell auf einem 286er mit 10 MHz und 2,5 Mbyte Speicher und mit einem halben Dutzend Sitzungen. Und es funktioniert besonders schnell, wenn Sie den Aufwand bedenken, den OS/2 betreiben muß, um vom Protected-Modus in den Real-Adress-Modus zu schalten. Dies ist genau das, was passiert, wenn Sie bei dem Taskwechsel über eine DOS-Sitzung wechseln.

Der Einsatz von OS/2

Nachdem Sie OS/2 installiert und laufen haben, können Sie einige Vorzüge gegenüber DOS erkunden. Wie Sie in der gesonderten OS/2-Kommandoübersicht sehen können, wurden einige von DOS übernommene Kommandos dadurch verbessert, daß sie mehrere Argumente akzeptieren. Des weiteren sind einige Substitutionsmöglichkeiten, die in DOS nur in BAT-Dateien funktionieren, unter OS/2 in der Kommandozeile möglich. Wenn eine Umgebung den Eintrag

```
LIB=C:\MSC\OS2\LIB
```

enthält, und Sie das Kommando

```
CD %LIB%
```

eingeben, dann wechseln Sie in das Unterverzeichnis \MSC\OS2\LIB auf dem Laufwerk C. Unter DOS war dies nur in Batch-Dateien möglich.

Ein weiterer Vorzug ist das DETACH-Kommando. DETACH läßt Sie von der Kommandozeile einen Hintergrundprozeß starten. Zum Beispiel startet die Kommandozeile

```
DIR | SORT > DSORT.TXT
```

einen Prozeß, der DIR ausführt, das Ergebnis über eine Pipe an SORT weitergibt, und dessen Ergebnis in eine Ausgabedatei schreibt.

```
DETACH DIR | SORT > DSORT.TXT
```

verlagert den ganzen Prozeß in eine Hintergrundsitzung. DETACH erwartet, daß der ganze Prozeß keine Tastatureingabe benötigt. Nach Erzeugen des Prozesses zeigt DETACH die Prozeß-ID an und kehrt zum startenden Prozeß, welcher in diesem Falle aus der OS/2-Kommandozeile besteht, zurück. So ist es leicht möglich größere Tasks in den Hintergrund zu verlagern und wie gewöhnlich weiterzumachen.

Ein weiterer Vorteil wird klar, wenn Sie ein OS/2-Kommando eingeben müssen, während Sie ein Programm laufen haben. Dies ist speziell dann vorteilhaft, wenn Sie einige Disketten formatieren müssen, oder Dateien löschen wollen, ohne das Programm zu verlassen. Das Formatieren arbeitet nach dem Start im Hintergrund weiter nachdem Sie [Alt][Esc] gedrückt haben, um in die ursprüngliche Sitzung zurückzuschalten. Wenn Ihnen dies gefällt, so sollten Sie gleich zu Beginn eine OS/2-Kommandozeilensitzung einrichten (oder RUN in die Datei CONFIG.OS2 schreiben, bevor der Programm-Selektor aufgerufen wird).

Ein weiterer Vorteil von OS/2 sind die verbesserten Möglichkeiten, um in der Kommandozeile komplexere Kommandos einzugeben, wie Sie auch in der gesonderten OS/2-Kommandoübersicht sehen.

Die Kommandozeile in *Abbildung 3* würde zum Beispiel eine sortierte Liste der Protected-Modus-Dienstprogramme erstellen, aber nur wenn der Laufwerks- und der Dateiwechsel erfolgreich waren. Sie können auch den ganzen Prozeß im Hintergrund laufen lassen, wenn Sie am Anfang das Kommando DETACH schreiben. Zusätzlich behält die Vordergrund-Sitzung das Original-Laufwerk und das Original-Verzeichnis bei, da der Wechsel nur im Hintergrundprozeß durchgeführt wird. Bedenken Sie, daß solche Veränderungen auf den Prozeß beschränkt bleiben.

```
c: && cd \os2\pbin && dir *.exe *.cmd | sort > tools.lst
```

Abbildung 3: *Unter OS/2 können mehrere Kommandos aus einer Kommandozeile ausgeführt werden.*

Schließlich werden Sie noch feststellen, daß die Möglichkeiten der Batchdateien von OS/2 gegenüber denjenigen von DOS verbessert wurden. Speziell die Kommandos SETLOCAL und ENDLOCAL tragen dazu bei. SETLOCAL merkt sich das aktuelle Laufwerk und das aktuelle Verzeichnis. ENDLOCAL stellt wieder den Zustand vor SETLOCAL ein. So können Sie also eine CMD-Datei wie die folgende schreiben:

```
@ECHO OFF
SETLOCAL
C: && CD\HA && HA
ENDLOCAL
```

Hierbei wird auf Laufwerk C in das Unterverzeichnis HA gewechselt und das Programm HA.EXE ausgeführt. Nach dem Programmende erhält die Sitzung wieder ihr altes Laufwerk und Unterverzeichnis. Diese Kommandos gleichen den UNIX-Kommandos PUSHDIR/POPDIR, die in einigen Systemen verfügbar sind.

Weniger Koffein

Der einzige Grund OS/2 nicht einzusetzen ist, daß es Kaffeepausen während längeren Kompilationsvorgängen überflüssig macht. Sie können immer etwas anderes tun! Da seine Multitasking-Fähigkeiten großartig sind, ist OS/2 ein produktives System, mit dem es Spaß macht, zu arbeiten. Wenn die richtigen Programme verwendet werden, die das System voll ausnutzen, werden viele Anwender auf OS/2 umsteigen. Dies ist für Sie ein guter Grund, schon jetzt OS/2-Programme zu entwickeln, falls Sie dies nicht schon begonnen haben.

Im nächsten Artikel dieser Serie werde ich die Konfiguration des C-Compilers behandeln und die notwendigen Dinge besprechen, die zum Erstellen des ersten Protected-Modus-Programms, einer Multithread-Version des »Hallo Welt«-Programms, erforderlich sind.

Richard Hale Shaw

OS/2-Programme mit mehreren Threads mit CodeView 2.2 debuggen:

Debuggen von Multithread-Programmen

Das Debuggen ist nie eine vergnügliche Angelegenheit, selbst wenn das zu untersuchende Programm relativ einfach ist. Aber stellen Sie sich die Fehlersuche in einem OS/2-Programm mit mehreren Threads vor. Alle Threads laufen im gleichen Programm parallel. Manchmal sind die Threads unabhängig voneinander aber oft haben sie auch zueinander Verbindung.

Zuerst erscheint der Gedanke, ein Programm mit mehreren Threads unter OS/2 zu debuggen, wie ein Alptraum, und manchmal ist es auch einer. Microsofts CodeView in der Version 2.2, wie er in den OS/2-Versionen der Sprachen von Microsoft (BASIC 6.0, C 5.1, FORTRAN 4.1, Macro Assembler 5.1 und Pascal 4.0) enthalten ist, bietet einige neue und verbesserte Kommandos, um die Fehlersuche in Programmen mit mehreren Threads zu unterstützen.

Obwohl ich einige Probleme hatte, ist die Version 2.2 von CodeView ein mächtiges Hilfsmittel für den OS/2-Programmierer. Wir sehen uns im folgenden diese neuen Möglichkeiten der Fehlersuche in mehreren Threads im Programm THREADS an und machen einige Debugger-Übungen.

Demonstration mehrerer Threads

Das Programm THREADS in *Listing 1* erzeugt vier weitere Threads. Jeder dieser vier Threads schreibt seine Thread-ID-Nummer an zufällige Positionen eines Quadranten des Bildschirms. Sie können THREADS durch Drücken der [Esc]-Taste verlassen.

THREADS benötigt die neuen Multithread-(MT)-Headerdateien und die Laufzeitbibliothek (LLIBCMT.LIB), die im Microsoft C-Compiler Version 5.1 enthalten sind. Alle C-Funktionen in dieser Bibliothek sind reentrant und können von mehreren Threads eines C-Programms ohne Probleme aufgerufen werden.

THREADS erzeugt die vier Threads in seiner Funktion `main`, die alle dieselbe Thread-Funktion, nämlich `ThreadFunction` benutzen. Anstatt `DosCreateThread` aufzurufen, um die Threads zu starten, bedient sich THREADS der Funktion `_beginthread`, einer neuen Funktion von C 5.1, die in C-Programmen benutzt werden muß, wenn die Multithread-Bibliothek verwendet wird.

Außer der Nutzung der Multithread-Bibliothek hat `_beginthread` für den Programmierer noch einen anderen Vorteil. Hierbei darf der Thread-Funktion ein Far-Zeiger übergeben werden. Im Falle von THREADS zeigt dieser Far-Zeiger auf die THREADPARAM-Struktur, die dazu dient, jedem Thread die Zeilen- und Spaltenangabe der Koordinaten seines Bildschirmquadranten mitzuteilen.

```
#-------------------
# THREADS make file
#-------------------

threads.obj : threads.c
     cl -c -Alfw -G2s -W3 -Zi threads.c

threads.exe : threads.obj threads.def
     link threads, /align:16, NUL, /nod llibcmt doscalls,
                   threads /CO
```

```
/*-----------------------------------------------------------
   THREADS.C -- Multi-Thread OS/2 Program for CodeView Demo
                Programmed by Charles Petzold, 6/88
  -----------------------------------------------------------*/

#define INCL_DOS
#define INCL_VIO
#define INCL_KBD

#include <os2.h>
#include <mt\process.h>
#include <mt\stdlib.h>

     /*--------------------------------------------
        Structure definition and function declarations
       ----------------------------------------------*/

typedef struct
     {
     SHORT xLeft ;
     SHORT yTop ;
     SHORT xRight ;
     SHORT yBottom ;
     }
     THREADPARAM ;

typedef THREADPARAM FAR *PTHREADPARAM ;

BOOL                CheckKey        (CHAR ch) ;
VOID _CDECL FAR ThreadFunction (PTHREADPARAM ptp) ;
VOID                WriteChar       (CHAR ch, USHORT usRow,
                                     USHORT usCol) ;
VOID PASCAL FAR ExitRoutine     (USHORT usTermCode) ;
VOID                ClearScreen     (BOOL fCursorOff) ;


     /*----------------
        main() function
       ----------------*/

INT _CDECL main (VOID)
     {
     static INT  iStack [4][2048] ;
     static CHAR szErrorMsg[] ="THREADS: Cannot start thread";
     KBDKEYINFO  kbci ;
     SHORT       i ;
     THREADPARAM tp[4] ;
     VIOMODEINFO viomi ;

          /*------------------------------------------
             Clear screen and get video mode information
            --------------------------------------------*/

     DosExitList (EXLST_ADD, ExitRoutine) ;
     ClearScreen (TRUE) ;

     viomi.cb = sizeof viomi ;
     VioGetMode (&viomi, 0) ;

          /*-------------------
             Start up four threads
            -------------------*/
```

Listing 1: *Das Programm THREADS.*

```
    for (i = 0 ; i < 4 ; i++)
         {
         tp[i].xLeft = i % 2 ? viomi.col / 2 : 0 ;
         tp[i].yTop  = i > 1 ? viomi.row / 2 : 0 ;

         tp[i].xRight  = tp[i].xLeft + viomi.col / 2 ;
         tp[i].yBottom = tp[i].yTop  + viomi.row / 2 ;

         if (-1 == _beginthread (ThreadFunction, iStack[i],
                                 sizeof iStack[i], tp + i))
              {
              VioWrtTTY (szErrorMsg, sizeof szErrorMsg - 1,0);
              return 1 ;
              }
         }

         /*--------------------------
            Wait for Esc key to exit
           --------------------------*/
    do
         KbdCharIn (&kbci, IO_WAIT, 0) ;
    while (!CheckKey (kbci.chChar)) ;

    return 0 ;
    }

     /*-----------------------------------------------------
        CheckKey() function -- Returns TRUE if ch is Escape
       -----------------------------------------------------*/

BOOL CheckKey (CHAR ch)
    {
    return ch == '\33' ;
    }

     /*------------------
        ThreadFunction()
       ------------------*/

VOID FAR _CDECL ThreadFunction (PTHREADPARAM ptp)
    {
    CHAR   ch ;
    SEL    selGlobal, selLocal ;
    TID    tid ;
    USHORT usRow, usCol ;

         /*----------------------------------------
            Get thread ID and convert to character
           ----------------------------------------*/

    DosGetInfoSeg (&selGlobal, &selLocal) ;
    tid = ((PLINFOSEG) MAKEP (selLocal, 0)) -> tidCurrent ;
    ch  = (CHAR) tid + '0' ;
    srand (tid) ;

         /*----------------------------------------
            Display thread ID in random locations
           ----------------------------------------*/
    while (TRUE)
         {
         usCol = ptp->xLeft + rand () % (ptp->xRight
                                        - ptp->xLeft) ;
         usRow = ptp->yTop  + rand () % (ptp->yBottom
                                        - ptp->yTop) ;
         WriteChar (ch, usRow, usCol) ;
         }
    }

     /*-------------------------------------------------------
        WriteChar() function -- Displays TID, pauses, then blank
       -------------------------------------------------------*/

VOID WriteChar (CHAR ch, USHORT usRow, USHORT usCol)
    {
    VioWrtCharStr (&ch, 1, usRow, usCol, 0) ;
    DosSleep (100L) ;
    VioWrtCharStr (" ", 1, usRow, usCol, 0) ;
    }
```

Listing 1: *(Fortsetzung)*

```
     /*-----------------------------------------------
        ExitRoutine() function -- Exit list processing
       -----------------------------------------------*/

VOID FAR PASCAL ExitRoutine (USHORT usTermCode)
    {
    usTermCode ;   /* to prevent compiler warning */

    ClearScreen (FALSE) ;
    DosExitList (EXLST_EXIT, NULL) ;
    }

     /*--------------------------------------------------------
        ClearScreen() function -- Also turns cursor off and on
       --------------------------------------------------------*/

VOID ClearScreen (BOOL fCursorOff)
    {
    USHORT          usAnsi ;
    VIOCURSORINFO vioci ;

    VioGetAnsi (&usAnsi, 0) ;
    VioSetAnsi (ANSI_ON, 0) ;
    VioWrtTTY ("\33[2J", 4, 0) ;
    VioSetAnsi (usAnsi, 0) ;

    VioGetCurType (&vioci, 0) ;
    vioci.attr = fCursorOff ? -1 : 0 ;
    VioSetCurType (&vioci, 0) ;
    }
```

```
;-----------------------------------
; THREADS.DEF module definition file
;-----------------------------------

NAME           THREADS  WINDOWCOMPAT

DESCRIPTION    'Multi-Thread Demo Program by Ch. Petzold, 1988'
PROTMODE
HEAPSIZE       1024
STACKSIZE      4096
```

Listing 1: *(Ende)*

Jeder Thread, der die Funktion `ThreadFunction` ausführt, erhält seine Thread-ID vom lokalen Informationssegment. Die Thread-ID ist in einer automatischen (Stack-) Variablen enthalten. Da jeder Thread seinen eigenen Stack besitzt, sind alle Stack-Variablen nur diesem Thread zugänglich. (Alle statischen Variablen in einer Funktion werden von allen Threads benutzt, die diese Funktion aufrufen).

Die Funktion `rand` wird benutzt, um die Thread-ID an eine zufällige Stelle innerhalb des Bildschirmquadrates zu schreiben. Die Funktion `rand` in der Bibliothek LLIBCMT.LIB erzeugt eine unabhängige Pseudo-Zufallszahlenfolge für jeden aufrufenden Thread. Im Programm THREADS ist dies nicht ganz das, was wir wollen. Um verschiedene Zufallszahlenfolgen für jeden Thread zu erreichen, setzt die Funktion `ThreadFunction` einen unterschiedlichen Startwert (durch den Aufruf von `srand`), der der Thread-ID entspricht.

Beachten Sie, daß die zwei Funktionen `CheckKey` und `WriteChar` nur von einer Stelle im Programm aufgerufen werden und keine eigenen Funktionen sein müssen. Dies wurde deshalb so ausgeführt, um Breakpunkte auf die Funktionen setzen zu können.

Die Makedatei von THREADS enthält den Schalter *-Zi* für die Kompilierung und den Schalter */CO* für den Linkvorgang. Diese beiden CodeView-Schalter übernehmen die Debugger-Informationen in die Datei THREADS.EXE.

Einfrieren und Auftauen

Wenn Sie ein Programm mit mehreren Threads debuggen, möchten Sie bestimmt auch nur einen Thread laufen lassen können, oder einen Thread stoppen, während die anderen laufen. Zu diesem Zweck verfügt CodeView über das Konzept des »Einfrierens« und »Auftauens« von Threads. Ein eingefrorener Thread läuft nicht. Threads werden durch Kommandos eingefroren und aufgetaut, die Sie im Dialogfenster am unteren Rand von CodeView eingeben. Andere Kommandos tauen Threads nur auf und frieren sie während der Zeit ein, in der das Kommando aktiv ist. Wir werden diese Kommandos kurz ansprechen.

Um das Programm THREAD.EXE in die OS/2-Version von CodeView zu laden, müssen Sie das Kommando

```
CVP THREADS
```

benutzen. Sie können den Schalter `/2` verwenden, um CodeView auf einem zweiten Monitor laufenzulassen, falls Sie einen besitzen.

Das Dialogfenster von CodeView zeigt folgenden Prompt:

```
001>
```

Dieser neue CodeView-Prompt ist die ID-Nummer des aktuellen Threads. Beim Start einer OS/2-Anwendung hat diese nur einen Thread, der die Nummer 1 hat. (Thread 1 besitzt einige besondere Eigenschaften. Er bearbeitet zum Beispiel die Signale, falls ein Signalhandler durch den Aufruf von `DosSetSigHandler` installiert ist.) Startet ein Programm weitere Threads, so bekommen diese der Reihenfolge nach die Nummern 2, 3, 4 und so weiter. Die Funktion `_beginthread` und die neue Multithread-Bibliothek erlauben es einem Programm, bis zu 32 Threads zu besitzen.

Probieren Sie das neue CodeView-Statuskommando:

```
001>~
```

Dieses Kommando zeigt alle Threads des Prozesses mit ihrem Status an. In diesem Fall wird nur der Thread 001 mit dem Zustand `Runnable` angezeigt.

Setzen Sie einen Breakpunkt auf die Funktion `CheckKey` und geben Sie das Kommando `Go` (G) ein.

```
001>BP CheckKey
001>G
```

Das Programm erzeugt die vier Threads, und diese laufen normal und zeigen ihre Thread-ID-Nummer an einer zufälligen Stelle des Bildschirms an.

```
 File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                              threads.c
87:
88:          /*-----------------------------------------------------
89:             CheckKey() function -- Returns TRUE if ch is Escape
90:            ------------------------------------------------------*/
91:
92:     BOOL CheckKey (CHAR ch)
93:          {
94:          return ch == '\33' ;
95:          }
96:
97:          /*------------------
98:             ThreadFunction()
99:            ------------------*/
100:
101:    VOID FAR _CDECL ThreadFunction (PTHREADPARAM ptp)
102:         {
005  Runnable     512
004  Runnable     512
003  Runnable     512
002  Runnable     512
001  Runnable     512
001>
```

Bild 1: *Das Statuskommando von CodeView zeigt den Status aller Threads eines Programms an.*

Unterbrechen Sie das Programm durch Druck auf die Leertaste. Da Sie einen Breakpunkt auf die Funktion `CheckKey` gesetzt haben, veranlaßt ein Tastendruck CodeView, am Beginn der Funktion zu stoppen. Geben Sie jetzt das Statuskommando erneut ein:

```
001>~
```

Sie sehen jetzt die Anzeige von *Bild 1.* Die Threads 001 bis 005 sind jetzt `Runnable`.

Verwenden Sie einen zweiten Monitor, so können Sie sehen, daß das ganze Programm unterbrochen ist, sobald CodeView die Kontrolle übernommen hat. (Verwenden Sie nur einen Monitor, so können Sie zu der Sitzung umschalten, in der THREADS läuft, um sich zu überzeugen, daß es wirklich unterbrochen ist.) Ein im Debugger laufendes Programm hält an, wenn CodeView die Kontrolle hat.

Versuchen Sie jetzt Thread 2 einzufrieren. Das Kommando dazu lautet:

```
001>~2F
```

Alle neuen Thread-Kommandos von CodeView beginnen mit der Tilde (~). Die `2` bezeichnet den Thread mit der ID 2. `F` steht für einfrieren (freeze). Sie können sehen, daß Thread 2 gestoppt ist, indem Sie das Statuskommando eingeben:

```
001>~
```

Thread 2 wird jetzt als `Frozen` (eingefroren) angezeigt. Geben Sie erneut das Go-Kommando ein

```
001>G
```

so sehen Sie, daß Thread 2 gestoppt ist.

Drücken Sie erneut die Leertaste, um bei der Funktion `CheckKey` zu stoppen. Sie können alle Threads stoppen, indem Sie einen Stern statt einer Zahl angeben:

```
001>~*F
```

Das Statuskommando

```
001>~
```

zeigt nun alle Threads als `Frozen` an. Lassen Sie uns wieder starten:

```
001>G
```

Kommando	wirkt auf
~	alle Threads
~n	Thread n
~.	aktuellen Thread
~#	zuletzt ausgeführten Thread
~*	alle Threads

Tabelle 1: *Die verschiedenen Varianten des Statuskommandos von CodeView.*

Stopp-Kommando	Start-Kommando	Zugehöriger Thread
~nF	~nU	Thread n
~.F	~.U	Aktueller Thread
~#F	~#U	Zuletzt ausgeführter Thread
~*F	~*U	Alle Threads

Tabelle 2: *Varianten der CodeView-Kommandos zum Einfrieren und Auftauen.*

Sie können sich vielleicht vorstellen, was das Kommando Go in dem Fall tut, in dem alle Threads gestoppt sind. Tatsächlich startet Go vorübergehend den aktuellen Thread, der wie es der Prompt zeigt, der Thread 001 ist, bevor es das Programm wieder startet. Sie können deshalb wieder mit der Leertaste anhalten.

Starten Sie jetzt den Thread 005 wieder, und geben Sie Go ein:

```
001>~5U
001>G
```

Das Kommando zum Auftauen ist `U` (unfreeze). Es benutzt dieselbe Syntax wie das Kommando zum Einfrieren. Nur der erste und der fünfte Thread laufen jetzt. Drücken Sie die Leertaste, starten alle Threads und geben Sie Go ein:

```
001>~*U
001>G
```

Alle Threads laufen jetzt normal. Drücken Sie die [Esc]-Taste, um das Programm THREADS zu beenden. CodeView hält bei der Funktion `CheckKey` an. Sie können das Programm normal beenden, indem Sie noch einmal Go eingeben:

```
001>G
```

und anschließend `Q` um CodeView zu verlassen:

```
001>Q
```

Kommandosyntax

Alle neuen Thread-Kommandos von CodeView beginnen mit einer Tilde (~). Das Statuskommando ist entweder eine Tilde allein, oder eine Tilde gefolgt von einer Zahl, einem Punkt, einem Doppelkreuz, oder einem Stern, wie in *Tabelle 1* gezeigt.

Die Kommandos ~ und ~* sind gleich. Sie zeigen alle aktiven Threads an. Sie können den Status eines Threads sehen, indem Sie nach der Tilde die Thread-ID-Nummer angeben. Der Punkt zeigt den Status des aktuellen Threads an, welches derjenige Thread ist, dessen ID im CodeView-Prompt angezeigt wird. Das Kommando ~# gibt den Status des zuletzt ausgeführten Threads an. Dies ist normalerweise der aktuelle Thread. Das muß aber nicht sein, wenn Sie den aktuellen Thread wechseln, ohne erneut einen Befehl auszuführen.

Die Kommandos zum Auftauen und Stoppen sind in *Tabelle 2* dargestellt. Sie erlauben das gezielte Einfrieren und Auftauen eines speziellen Threads, des aktuellen Threads, des zuletzt ausgeführten Threads oder aller Threads.

Breakpunkte

In dem Beispiel mit den Kommandos zum Einfrieren und zum Auftauen benutzten wir das normale BP-Kommando, um einen Breakpunkt auf die Funktion `CheckKey` zu setzen. Die `CheckKey`-Funktion wird nur vom Thread 1 aufgerufen, so daß es keinen Zweifel gibt, wie CodeView dieses Kommando interpretiert. Wie sieht es aber aus, wenn Sie einen Breakpunkt in einer Befehlszeile setzen wollen, die von mehreren Threads ausgeführt wird (wie in der Funktion `ThreadFunction`)? Ist der Breakpunkt nur für einen oder für alle Threads gültig? Die Antwort lautet: So, wie Sie es wünschen.

Laden Sie THREADS wieder in CodeView:

```
CVP THREADS
```

und setzen Sie einen Breakpunkt auf die Funktion `CheckKey`:

```
001>BP CheckKey
```

Setzen Sie auch einen Breakpunkt auf die Funktion `ThreadFunction`:

```
001>BP ThreadFunction
```

Das Kommando BP allein trifft für alle Threads zu. Sie können das leicht durch die Eingabe des Kommandos Go überprüfen:

```
001>G
```

CodeView übergibt Ihnen wieder die Kontrolle, wenn ein Thread (möglicherweise, aber nicht notwendigerweise Thread 2) startet und den Beginn der Thread-Funktion erreicht. (Obwohl CodeView am Beginn der Funktion `ThreadFunction` unterbricht, bedeutet dies nicht, daß der Thread 2 bis jetzt noch keinen Befehl abgearbeitet hat. Bevor er die Funktion `ThreadFunction` erreicht, hat er schon einige Programmteile in der Funktion `_beginthread` hinter sich.)

Wenn CodeView Ihnen die Kontrolle wieder übergibt, so sehen Sie einen Prompt, in dem Thread 2 (oder welcher auch immer) als aktuell angezeigt wird:

```
002>
```

Die Anzeige im Programmfenster von CodeView zeigt den Cursor am Anfang der Funktion ThreadFunction, wie dies im *Bild 2* zu sehen ist.

Sie können weitermachen, indem Sie das Go-Kommando eingeben:

```
002>G
```

Nun erreicht Thread 3 die Funktion ThreadFunction. Starten Sie erneut durch das Go-Kommando:

```
003>G
```

Thread 4 bringt das Programm zum Stoppen. Geben Sie erneut ein Go-Kommando ein:

```
004>G
```

Thread 5 ist jetzt an der Reihe. Und schließlich starten Sie noch einmal mit dem Go-Kommando:

```
005>G
```

Jetzt laufen alle 5 Threads des Programms normal.

Drücken Sie die Leertaste, um Thread 1 an der Funktion CheckKey zu stoppen. Setzen Sie jetzt einen Breakpunkt auf die Funktion WriteChar. Obwohl die Funktion WriteChar von den Threads 2 bis 5 aufgerufen wird, bezieht sich dieser Breakpunkt nur auf den Thread 5:

```
001>~5BP WriteChar
```

Wie alle Thread-spezifischen Kommandos beginnt auch dieses mit einer Tilde. Darauf folgt die Thread-ID-Nummer und BP, das normale Breakpunkt-Kommando. Starten Sie wieder:

```
001>G
```

Jetzt stoppt Thread 5 am Beginn der Funktion WriteChar. Das Programmfenster zeigt den Cursor am Beginn der Funktion WriteChar und der Prompt gibt den Thread 5 an:

```
005>
```

Sie können eine Liste aller Breakpunkte mit dem Kommando BL sehen:

```
005>BL
```

CodeView zeigt die Informationen wie in *Bild 3* dargestellt. Die Aufstellung zeigt eine Tilde und die Thread-ID aller Breakpunkte, die sich nur auf einen bestimmten Thread beziehen (in diesem Fall Breakpunkt 2). Ein Breakpunkt, der sich auf alle Threads bezieht, wie etwa die Breakpunkte 0 und 1, besitzt diese Information nicht.

Löschen Sie den Breakpunkt 2 und setzen Sie einen neuen, der wirksam wird, wenn der Thread 5 die Funktion WriteChar hundertmal ausgeführt hat:

```
005>BC 2
005>~5BP WriteChar 100
005>G
```

Dieses Programm läuft einige Zeit (langsamer als gewöhnlich) und hält dann an der Funktion WriteChar an. Löschen Sie den Breakpunkt und setzen Sie einen, der beim hundertsten Ausführen von WriteChar aktiv wird, egal von welchem Thread die Funktion aufgerufen wird:

```
005>BC 2
005>BP WriteChar 100
005>G
```

Kommando	Zugehöriger Thread
BP	Alle Threads
~nBP	Thread n
~.BP	Aktueller Thread
~#BP	Zuletzt ausgeführter Thread
~*BP	Alle Threads

Tabelle 3: *Varianten der Breakpunkt-Kommandos von CodeView.*

Jetzt gibt CodeView die Kontrolle schneller zurück, da die Anzahl aller Threads erhöht wird, nicht nur von Thread 5. Beim Erreichen des Breakpunkts kann der aktuelle Thread 2, 3, 4 oder 5 sein (ich benutze deswegen ein Fragezeichen in der Anzeige):

Sie können alle Breakpunkte löschen und das Programm erneut starten:

```
00?> BC*
00?>G
```

Drücken Sie die [Esc]-Taste, um das Programm zu beenden, und beenden Sie CodeView mit dem Quit-Kommando:

```
001>Q
```

Das Breakpunkt-Kommando

Tabelle 3 zeigt die verschiedenen Breakpunkt-Kommandos von CodeView. Die Kommandos BP und ~*BP sind gleich und betreffen alle Threads. Wenn Sie also zum Beispiel auf die Funktion CheckKey einen Breakpunkt setzen, so können Sie eines der folgenden Kommandos eingeben:

```
001>BP CheckKey
001>~*BP CheckKey
```

Da nur Thread 1 die CheckKey-Funktion aufruft, können Sie auch folgendes eingeben:

```
001>~1BP CheckKey
```

Ist Thread 1 der aktuelle (wie in diesem Fall am Prompt zu sehen), kann das Kommando auch so aussehen:

```
001>~.BP CheckKey
```

Sie können auch einen Breakpunkt für einen noch nicht existierenden Thread setzen. Wenn Sie zum Beispiel THREADS neu laden, können Sie eingeben:

```
001>~5BP ThreadFunction
```

CodeView stoppt nur, wenn Thread 5 die ThreadFunction ausführt.

Sie können Zeilennummern oder absolute Adressen mit allen Varianten des Breakpunkt-Kommandos verwenden. Sie können auch den Durchlaufzähler und die anderen Kommandooptionen des BP-Kommandos verwenden. Wenn Sie einen Breakpunkt mit der Maus setzen (indem Sie die Zeile anklicken), oder mit der Taste [F9], so bezieht sich dieser Breakpunkt auf alle Threads. Nur beim BP-Kommando, das Sie im Dialogfenster eingeben, können Sie Breakpunkte für einzelne Threads angeben.

Kommando	stoppt	startet
G	-	aktuellen Thread
~nG	alle Threads, außer n	Thread n
~.G	alle Threads, außer den aktuellen	aktuellen Thread
~#G	alle Threads, außer den zuletzt ausgeführten	zuletzt ausgeführten Thread
~*G(+)	alle Threads, außer den aktuellen	aktuellen Thread
(+) Dies ist ein Fehler. Das Kommando ~*G sollte gleich dem Kommando G sein.		

Tabelle 4: *Varianten des Go-Kommandos von CodeView.*

Das Go-Kommando

Wir haben das Go-Kommando in den Übungen zusammen mit dem Auftauen, dem Einfrieren und den Breakpunkten verwendet, und es funktionierte wie wir das erwartet haben. Das Go-Kommando erlaubt es dem Programm einfach zu laufen. Die Kontrolle geht beim Erreichen eines Breakpunkts wieder an CodeView zurück. Die einzige Besonderheit trat im Beispiel zum Auftauen und Einfrieren, beim Einfrieren aller Threads auf, als wir entdeckten, daß das Go-Kommando es dem Thread 1 erlaubte, weiter zu laufen.

Tabelle 4 zeigt die Varianten des Go-Kommandos von CodeView. Wie Sie in der letzten Spalte von Tabelle 4 sehen, taut das Go-Kommando einen Thread zeitweise auf, wenn dieser eingefroren ist. Geht die Kontrolle an CodeView über, so erhält der Thread seinen früheren Status zurück. Wenn Sie die Thread-spezifischen Versionen des Go-Kommandos benutzen (~nG, ~.G oder ~#G), so friert CodeView zeitweise alle Threads ein, außer den im Kommando angegebenen. Dies erlaubt es Ihnen, die Ausführung eines einzigen Threads zu kontrollieren, während die anderen von der Ausführung abgehalten werden.

Lassen Sie uns eine kleine Übung durchführen, um zu sehen, wie dies funktioniert. Laden Sie THREADS:

```
CVP THREADS
```

Setzen Sie einen Breakpunkt auf die Funktion `WriteChar`, der nur Thread 5 betrifft, und starten Sie:

```
001>~5BP WriteChar
001>G
```

CodeView unterbricht, wenn Thread 5 die Funktion `WriteChar` erreicht. Der Prompt zeigt Thread 5 als aktuellen Thread an.

```
005>
```

Sie können einige weitere Go-Kommandos eingeben und sehen, daß alle Threads zwischen den Unterbrechungen laufen.

```
005>G
```

Versuchen Sie jetzt ein Go-Kommando, das nur Thread 5 betrifft:

```
005>~5G
```

In diesem Falle läuft nur der Thread 5. CodeView stoppt alle anderen Threads, bevor es das Programm wieder startet. Löschen Sie den Breakpunkt und starten Sie wieder:

```
005>BC*
005>G
```

Beenden Sie das Programm durch Druck auf die [Esc]-Taste und beenden Sie CodeView:

```
005>Q
```

Das Go-Kommando aus dem Menü oder die Taste [F5] funktionieren ebenso wie die Eingabe von G im Dialogfenster. Hält CodeView das Programm an, ist aber nicht unmittelbar ersichtlich, welcher Thread die Unterbrechung verursacht hat. Deshalb ist es oftmals besser, beim Debuggen mit mehreren Threads das Go-Kommando im Dialogfenster zu benutzen

Breakpunkte und Einzelschritte

Bevor wir fortfahren, ist es hilfreich, die Debugging-Fähigkeiten des 80286-Mikroprozessors zu untersuchen. Ein Debugger kann einen Breakpunkt auf einen Maschinenbefehl setzen, indem er das erste Byte des Befehles durch CCh ersetzt. Diese Anweisung löst einen Interrupt 3 aus. Führt das Programm diese Anweisung aus, gibt der Interrupt die Kontrolle an den Debugger zurück. Der Debugger kann dann das Originalbyte wieder zurückschreiben, bevor er das Programm weiter ausführt.

Ein Debugger kann ein Programm in Einzelschritten ausführen, indem er das Trap-Flag setzt. Dann wird nach jeder Anweisung ein Interrupt 2 ausgeführt. In OS/2 setzt ein Debugger nicht selbst das Trap-Flag. Dies wird von der OS/2 Debug-Funktion `DosPTrace` erledigt. Ebenso wird beim Auftreten eines Interrupts 2 oder 3 die Kontrolle an OS/2 übergeben, welches dann von einem früher vom Debugger aufgerufenen `DosPTrace`-Aufruf zurückkehrt. `DosPTrace` ist im Programmer's Reference Manual des OS/2 Programmer's Toolkit dokumentiert.

Wenn Sie mit dem BP-Kommando einen Breakpunkt setzen, so setzt CodeView den Breakpunkt-Interrupt in Ihr Programm. Wenn Sie ein Programm mit den Befehlen T (Trace), P (Einzelschritt, program step) oder E (Ausführung, execute) tracen, werden Sie vermuten, daß CodeView die Einzelschrittfähigkeit benutzt. Dies ist aber nicht so. Wenn Sie Quellcode-Zeilen tracen, setzt CodeView einen Breakpunkt auf die letzte Maschinenanweisung, die zu diesem Quellcode-Statement gehört. Wenn Sie das P-Kommando verwenden, um im Assemblermodus zu tracen, benützt CodeView die Breakpunkt-Fähigkeit, um Funktionsaufrufe zu übergehen.

Aus diesem Grunde arbeiten die Kommandos Trace, Einzelschritt (single step) und Ausführen (execute) im Quellcode- und im Assemblermodus etwas unterschiedlich.

Kommando	stoppt	startet	übergibt Kontrolle an
T	-	aktuellen Thread	nicht vorhersagbar
~nT	alle Threads, außer n	Thread n	nächsten Befehl des Thread n
~nT	alle Threads, außer n	aktuellen Thread	nächsten Befehl des aktuellen Threads
~#T	alle Threads, außer den zuletzt ausgeführten	zuletzt ausgeführten Thread	nächsten Befehl des zuletzt ausgeführten Threads
~*T(+)	alle Threads, außer dem aktuellen	aktuellen Thread	nächsten Befehl des aktuellen Threads

(+) Dies ist ein Fehler. Das Kommando ~*T sollte mit dem Kommando T identisch sein.

Tabelle 5: *Varianten des Trace-Kommandos im Quellcode-Modus. (Die Kommandos P und E arbeiten ähnlich.)*

Kommando	Stoppt Thread	Startet Thread	Übergibt Kontrolle an
T	Alle Threads, außer dem Aktuellen	Aktuellen Thread	Nächsten Befehl des aktuellen Threads
~nT	Alle Threads, außer n	Thread n	Nächsten Befehl des Threads n
~.T	Alle Threads, außer den Aktuellen	Aktuellen Thread	Nächsten Befehl im aktuellen Thread
~#T	Alle Threads, außer den zuletzt ausgeführten	Zuletzt ausgeführten Thread	Nächsten Befehl im zuletzt ausgeführten Thread
~*T(+)	Alle Threads, außer dem Aktuellen	Aktuellen Thread	Nächsten Befehl im aktuellen Thread

(+) Dies ist ein Fehler. Das Kommando ~*T sollte im Assembler- oder im gemischten Modus nicht möglich sein.

Tabelle 6: *Variationen des Trace-Kommandos im Assembler-Modus oder gemischten Modus. (P und E arbeiten ähnlich.)*

Tracen

Wir wollen THREADS noch einmal für ein Trace-Beispiel unter CodeView laden: `CVP THREADS`

Setzen Sie einen Breakpunkt auf die Funktion `ThreadFunction`, der nur Thread 5 betrifft, und starten Sie:

```
001>~5BP ThreadFunction
001>G
```

```
File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                              threads.c
96:
97:          /*-----------------
98:             ThreadFunction()
99:            -----------------*/
100:
101:     VOID FAR _CDECL ThreadFunction (PTHREADPARAM ptp)
102:          {
103:          CHAR   ch ;
104:          SEL    selGlobal, selLocal ;
105:          TID    tid ;
106:          USHORT usRow, usCol ;
107:
108:               /*----------------------------------------
109:                  Get thread ID and convert to character
110:                 ----------------------------------------*/
111:
112:          DosGetInfoSeg (&selGlobal, &selLocal) ;
113:          tid = ((PLINFOSEG) MAKEP (selLocal, 0)) -> tidCurrent ;
001>bp CheckKey
001>bp ThreadFunction
001>g
002>
```

Bild 2: *Wenn am Anfang einer Thread-Funktion ein Breakpunkt gesetzt wird, hält CoeView an, bevor Code aus dem Thread ausgeführt wird.*

```
File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                              threads.c
128:
129:          /*------------------------------------------------------------
130:             WriteChar() function -- Displays TID, pauses, then blank
131:            ------------------------------------------------------------*/
132:
133:     VOID WriteChar (CHAR ch, USHORT usRow, USHORT usCol)
134:          {
135:          VioWrtCharStr (&ch, 1, usRow, usCol, 0) ;
136:          DosSleep (100L) ;
137:          VioWrtCharStr (" ", 1, usRow, usCol, 0) ;
138:          }
139:
140:          /*------------------------------------------------
141:             ExitRoutine() function -- Exit list processing
142:            ------------------------------------------------*/
143:
144:     VOID FAR PASCAL ExitRoutine (USHORT usTermCode)
005>BL
0 e       0027:010D  _CheckKey:93
1 e       0027:0122  _ThreadFunction:102
2 e ~005 0027:01A4  _WriteChar:134
005>
```

Bild 3: *Mit dem Befehl zum Auflisten der Breakpunkte wird auch angezeigt, wenn ein Breakpunkt nur für einen Thread gültig ist.*

Sie sollten zu dieser Zeit noch im Quellcode-Modus sein. Zum Tracen einer Quellcodezeile können Sie folgendes Kommando einige Male eingeben:

```
005>~.T
```

Bei jeder Eingabe dieses Kommandos erlaubt CodeView die Ausführung einer Quellcode-Anweisung. Sie können auch

```
005>~.T
```

eingeben. Der Punkt steht für den aktuellen Thread, der hier Thread Nummer 5 ist.

Sie haben sicher auch bemerkt, daß keiner der anderen Threads bei der Eingabe dieses Kommandos läuft. Wenn Sie mit den Kommandos ~.T oder ~nT einen einzelnen Thread tracen, stoppt CodeView alle anderen Threads. Dies ist sowohl im Quellcode-, als auch im Assemblermodus so.

Das schwierigste Kommando ist T selbst, das im Quellcode- und im Assemblermodus unterschiedlich arbeitet. Um dies zu erläutern, schalten Sie bitte zuerst in den gemischten Modus:

```
005>S&
```

Geben Sie nun einige T-Kommandos ein:

```
005>T
```

An den Stellen mit Aufrufen der OS/2- oder Bibliotheksfunktionen sollten Sie das Kommando P verwenden, um das Durchlaufen der Funktionen zu vermeiden.

Beachten Sie, daß keiner der anderen Threads läuft, während Sie tracen. Das T-Kommando im Assembler-Modus oder im gemischten Modus ist identisch mit ~.T.

Schalten Sie jetzt in den Quellcode-Modus zurück:

```
005>S+
```

und geben Sie einige T-Befehle ein:

```
005>T
```

Im Quellcode-Modus erlaubt der T-Befehl den anderen Threads auch zu laufen. CodeView setzt auf den letzten Maschinensprachebefehl der Anweisung einen Breakpunkt und läßt das Programm laufen.

CodeView überprüft aber nicht, welcher Thread den Breakpunkt erreicht. In dem Programm THREADS führen die Threads 2 bis 5 die gleichen Anweisungen aus. Wenn Sie also die T-Anweisung im Quellcode-Modus verwenden, sehen Sie, daß die Programmausführung oft durch unterschiedliche Threads gestoppt wird. Beachten Sie dies, wenn Sie ein Programm debuggen, dessen Code von mehreren Threads gemeinsam benutzt wird. Sie können auch die Kommandos ~.T oder ~nT benutzen, um den Wechsel der Threads zu vermeiden.

Zum Beenden dieser Übung geben Sie bitte Go ein:

```
00?>G
```

Drücken Sie jetzt noch die [Esc]-Taste und geben Sie Q zum Verlassen von CodeView ein:

```
001>Q
```

Kommando	Aktueller Thread wird
~nS	Thread n
~.S	aktueller Thread
~#S	zuletzt ausgeführter Thread

Tabelle 7: *Variationen des Select-Kommandos von CodeView.*

Die Syntax der verschiedenen Trace-, Einzelschritt- und Ausführen-Kommandos ist in den *Tabellen 5* und *6* dargestellt.

Das Aktivieren der Trace- und Einzelschritt-Kommandos vom Menü oder das Benutzen der Funktionstasten [F8] oder [F10] ist identisch mit der Eingabe von T.

Wenn Sie feststellen, daß der aktive Thread nicht derjenige ist, den Sie benötigen, so können Sie dies mit dem Select-Kommando ~nS von CodeView ändern. Diese und andere Arten des Select-Kommandos sehen Sie in *Tabelle 7*.

Wie sie gesehen haben, darf bei allen Thread-spezifischen Kommandos das Nummern-Zeichen (#) für den zuletzt aktiven Thread eingegeben werden. Dies ist immer auch der aktive Thread, außer beim Wechsel mit dem Select-Kommando. Der zuletzt ausgeführte Thread wird vom Select-Kommando nicht geändert, sondern nur durch die Befehle Go, Trace, Einzelschritt oder Ausführen. In diesem Fall ist der zuletzt ausgeführte Thread der aktuelle Thread, nachdem die Kontrolle an CodeView zurückgegeben wurde.

Fenster und Threads

Obwohl die neuen Kommandos von CodeView beim Debuggen von Anwendungen mit mehreren Threads sehr hilfreich sind, sind sie doch nicht ganz befriedigend. Zum ersten können sie zum großen Teil nur im Dialogfenster eingegeben werden. Die Verwendung eines einzelnen Anzeigebildes, das umgeschaltet wird, ist ebenso keine gute Darstellung eines Programms mit vielen Threads.

Wenn CodeView sich weiterentwickelt, um den komplexen Architekturen von OS/2-Anwendungen gerecht zu werden, so könnte ein Interface mit mehreren Fenstern, wobei jedes für einen Thread vorhanden ist, geeigneter sein. Der Presentation Manager von OS/2 wäre die ideale Umgebung für solch einen Debugger.

Charles Petzold

Termine ... Termine ... Termine ... Termine

Mit Microsoft-Seminaren sicher in die Zukunft

Die ständige Leistungssteigerung in der Hard- und Software-Entwicklung läßt den Fachkräften der Industrie kaum eine Chance, die Möglichkeiten dieser Technologie in gleicher Schnelligkeit umzusetzen. Dabei ist es von sehr großer Bedeutung, Arbeitsbereiche mit Hilfe der EDV zu rationalisieren. Um die Integration der EDV und damit eine Arbeitserleichterung im betriebswirtschaftlichen Alltag zu erreichen, wird Microsoft die Institut-Seminare in München und Düsseldorf fortsetzen.

Diese Spezialseminare des Microsoft Instituts vermitteln in kleinen Gruppen intensiv all das, was zum Einstieg in die Programmentwicklung nötig ist. Modernste Trainingsmethoden sowie PC-Demonstrationen und -Übungen sind selbstverständlich. Die Dozenten befassen sich auch im persönlichen Gespräch ausführlich mit den individuellen Forderungen und Problemen der Teilnehmer. So bekommen professionelle Entwickler durch professionelle Schulung die Möglichkeit, ihren hohen Wissenstand den neuen Gegebenheiten anzupassen. Das Microsoft Institut bietet Seminare in drei Bereichen an:

Systemsoftware

Diese Seminare sind für PC-Software-Entwickler bestimmt. Die angebotenen Themen:

- MS-OS/2
- MS-Windows für Software-Entwickler
- MS-LAN Manager für Software-Entwickler

Applikationen

Diese Seminare sind in der ersten Linie für die Mitarbeiter des EDV-Benutzer-Services bestimmt. Die angebotenen Themen:

- MS-Word
- MS-Excel und MS-Excel Makro-Programmierung
- MS-Multiplan
- MS-Chart

Informationsseminare

Diese Seminare sind für die Mitarbeiter des EDV-Benutzer-Service bestimmt. Die angebotenen Themen:

- Einblick in Microsoft System-Software
- LAN Manager und SQL Server

Das Microsoft OS/2 Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Neben einem Überblick über den Intel 80286-Prozessor wird dann als Schwerpunkt dieses Seminars auf das Application Program Interface (API) eingegangen. In diesem Zusammenhang werden die Funktionen für Multitasking, Memory Management und Dynamic Linking ausführlich behandelt. Weitere Themen: Geräte- und Datei-Ein-/-Ausgabe. Die in diesem Seminar gezeigten Programmbeispiele sind in Microsoft C geschrieben.

Ort	Datum	Tag
Düsseldorf	19./20.06.89	Mo./Di.
München	29./30.05.89	Mo./Di.

Der Microsoft OS/2 Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das MS-OS/2-Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC den komfortablen Editor zu nutzen, Family-API-Programme zu schreiben und Device-I/O-Routinen zu erstellen sowie Multitasking-Funktionen zu nutzen und eigene Dynamic-Link-Bibliotheken zu erstellen; außerdem können sie die erweiterten Speicherverwaltungsmöglichkeiten des Intel 80286 nutzen und mit Hilfe des MS-OS/2 Memory Managers programmieren. Dieses Seminar ist übrigens nicht im SDK-Preis enthalten.

Ort	Datum	Tag
Düsseldorf	21./22./23.06.89	Mi./Do./Fr.
München	31.5./1./2.6.89	Mi./Do./Fr.

Das Microsoft Windows Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Dieses Seminar behandelt die Microsoft Windows-Benutzerschnittstelle, die Microsoft Windows-Systemarchitektur, die Programmierwerkzeuge in der Praxis und Erstellung von Microsoft Windows-Programmen an Beispielen. Dieses Seminar ist nicht im SDK-Preis enthalten.

Ort	Datum	Tag
Düsseldorf	26./27.06.89	Mo./Di.
München	05./06.06.89	Mo./Di.

Der Microsoft Windows Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das Microsoft Windows Einführungsseminar besucht haben.

Termine ... Termine ... Termine ... Termine

In diesem Seminar erwirbt der Teilnehmer durch praktische Tätigkeit an einem PC die Fertigkeiten zur Programmierung unter Microsoft Windows. Dazu werden verschiedene Aufgaben gestellt und Übungen abgehalten. Als Hauptthemen dieses Seminars werden Übungen zu dem »Windowing«, der grafischen Programmierschnittstelle, den Benutzerschnittstellen und dem Dynamic Linking durchgeführt. Dieses Seminar ist nicht im SDK-Preis enthalten.

Ort	Datum	Tag
Düsseldorf	28./29./30.06.89	Mi./Do./Fr.
München	07./08./09.06.89	Mi./Do./Fr.

Microsoft Presentation Manager Einführungs-Seminar

Dieses eintägige Seminar erleichtert dem Windows-Programmierer das Umsteigen von Microsoft Windows auf den Microsoft OS/2 Presentation Manager.

Am Anfang des Seminars wird eine sehr kurze Einführung in das Betriebssystem Microsoft OS/2 gegeben. Danach werden die konzeptionellen Unterschiede in beiden Systemen aufgezeigt und schließlich wird anhand von Programmbeispielen die Programmierung dieser Unterschiede erläutert.

Ort	Datum	Tag
Düsseldorf	19.05.89	Fr.
München	12.06.89	Di.

Microsoft LAN Manager für Windows-Programmierer

Dieses eintägige Seminar für Netzanwender und Netzwerk-Softwareentwickler mit Programmiererfahrung in einer höheren Programmiersprache vermittelt das neue zukunftsweisende Programmierkonzept, um den MS OS/2 LAN Manager bedienen zu können.

Das Seminar gibt zuerst einen Überblick über die Komponenten von MS OS/2. Dazu gehören Presentation Manager und LAN Manager. Anschließend werden die neuen Konzepte des LAN Managers erörtert, sowie neue und zukunftsweisende Programmiermethoden (hier ist die »distributed intelligence Method« zu nennen). Weiterhin gibt das Seminar in seinem letzten Abschnitt eine Bedienungsanleitung für den LAN Manager und vermittelt so einen »roten Faden« für den Netzwerkanwender.

Ort	Datum	Tag
Düsseldorf	16.05.89	Fr.
München	13.06.89	Di.

Microsoft LAN Manager und SQL Server Informationsseminare

Dieses eintägige Informationsseminar wendet sich an Vertriebs- und Supportmitarbeiter sowie Entscheidungsträger, die Gerätekonfigurationen für Arbeitsgruppen festlegen.

Die Teilnehmer gewinnen einen Überblick über die Architektur des Client-Server-Modells und die darin angelegten, neuen Möglichkeiten für die Gruppenarbeit in einem Netz durch gemeinsamen Zugriff auf eine zentrale Datenbank.

Dargelegt wird, was den LAN Manager von einem herkömmlichen Netz unterscheidet und welche Vorteile für den Anwender damit verbunden sind.

Die neuartige Architektur des SQL-Servers wird erläutert und welche zum Teil einmaligen, neuen Möglichkeiten sich in Hinsicht auf die Sicherheit, Datenintegrität, Transaktionen und Performance als Plattform für eigene Anwendungen und Verfügbarkeit ergeben. Schließlich werden die Oberflächen des LAN-Managers und des SQL-Servers erklärt und an Beispielen gezeigt, wie man mit dem System arbeitet und welche komplexen Abfragen SQL ermöglicht.

Ort	Datum	Tag
Düsseldorf	22.05.89	Mo.
München	21.06.89	Mi.

Microsoft Excel Makro-Programmierung

Dieses dreitägige Seminar wendet sich an PC-Software-Entwickler, die mit Microsoft Excel schon einigermaßen vertraut sind.

Das Seminar gibt anfangs einen grundsätzlichen Überblick über Microsoft Excel, der für das Verständnis der Makroprogrammierung nötig ist. Durch einfachere Einzelbeispiele wird der Teilnehmer zunächst mit der Makroprogrammierung vertraut gemacht. Es folgt der Einstieg in komplexe Problemlösungen mit Makros, in denen u.a. sowohl selbstdefinierte Menüs und Dialogboxen als auch Funktions- und Befehlsmakros eingebunden sind. Weiterhin geht das Seminar auf die Microsoft Excel-Programmierumgebung ein, wobei auf die Möglichkeiten hingewiesen wird, Informationen aus anderer Software mit Makros zu verarbeiten (Datev Datensätze) und wie mit Hilfe von Makrobefehlen die DDE-Schnittstelle von Microsoft Windows genutzt werden kann.

Ort	Datum	Tag
Düsseldorf	09./10./11.05.89	Di./Mi./Do.
	13./14./15.06.89	Di./Mi./Do.
München	06./07./08.06.89	Di./Mi./Do.

Mitteilungen .... Mitteilungen .... Mitteilungen

Neue Windows-Versionen von Microsoft

Windows/286 Version 2.1 und Windows/386 Version 2.1 sind die Bezeichnungen für zwei neue, noch leistungsfähigere Versionen der grafischen Betriebssystemerweiterung Windows, die Microsoft ab sofort anbietet.

»Die beiden neuen Versionen sind die aktuellsten Meilensteine, die die langfristige Weiterentwicklung und das Engagement von Microsoft für die Windows-Betriebssystem-Erweiterung markieren«, so Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH. Die neuen Windows-Versionen wurden für Anwender von PCs auf 80286- und 80386-Basis sowie für Benutzer schneller XT-Kompatibler entwickelt, die damit künftig alle Möglichkeiten, die die neuen Windows-Anwendungen bieten, nutzen können.

Herausragendes Merkmal der neuen Windows-Versionen ist die gesteigerte Arbeitsgeschwindigkeit, die durch eine Erhöhung des direkt adressierbaren MS-DOS-Arbeitsspeichers in AT- und AT-kompatiblen PCs von 640 auf 685 Kbyte erreicht wurde. Bei PS/2-Computern steht dieser erweiterte Speicher schon bei einer Hauptspeicher-Kapazität von 1 Mbyte zur Verfügung. Durch die damit erreichte Verringerung der Festplatten-Swapping-Aktivitäten können Windows-Programme um bis zu 50% schneller als bisher ablaufen.

Verbessert wurde auch das Installationsprogramm. Es erkennt jetzt automatisch die Hardware-Konfiguration des Anwenders, wie Bildschirm, Grafikkarte und Speichertyp (Expanded/Extended Memory).

Dieser »intelligente« Installationsvorgang gewinnt vor allem mit Blick auf eine offene PC-Umgebung an Bedeutung: Er bietet die Voraussetzungen für den Einsatz einer breiten Palette von neuen Drucker-, Plotter- und Bildschirmtreibern (u.a. HP-Deskjet und Paint-Jet, NEC P6/P7, Compaq-Plasma-Display).

Neben den genannten zeichnen sich die neuesten Versionen von Windows/286 und Windows/386 noch durch weitere Verbesserungen aus:

- Unterstützung von 131 Druckern und Plottern statt bisher 62;
- Unterstützung des hochauflösenden Farbgrafikmodus 8514 (Windows/286);
- Unterstützung von Herkules- und Chips & Technologies' 441-VGA-Modus;
- Optimale Zusammenarbeit mit MS-DOS 4.0;
- Aktualisierter AST-RAM-Page-Support;
- Aktualisierter Intel-Above-Board-Support gemäß den vollständigen EEM-Spezifikationen;
- Unterstützung der Bildschirme IBM-8514, Wyse700 und »Genius« von Micro Display Systems.

Zusätzlich zu den neuen Microsoft Windows-Versionen für Endanwender gibt es für Windows-Programmierer ein neues Windows Software Development Kit (SDK), Version 2.1. Es enthält leistungsfähige Programmierwerkzeuge zur Entwicklung grafikorientierter Windows-Anwendungen, die nach geringfügigen Modifikationen auch unter dem MS OS/2 Presentation Manager laufen können.

Die SDK-Tools beinhalten:

- Microsoft CodeView für Windows, eine spezielle Version des weit verbreiteten Microsoft Quellcode-Debuggers, der die Erstellung von Windows-Code erheblich erleichtert;
- einen Dialog-Editor zur Entwicklung von Dialog-Boxen;
- einen Icon-Editor zur Entwicklung spezieller Symbole, Cursor und Bitmaps;
- das automatische Wartungsprogramm MAKE;
- umfassende Handbücher, einschließlich eines »Programmer's Reference Manual«, eines »Learning Guide« und eines »Application Style Guide«.

Das SDK bietet die Voraussetzung, auch Anwendungsprogramme für den MS OS/2 Presentation Manager zu entwickeln, da MS Windows und der MS Presentation Manager auf denselben Programmiermodellen und Entwicklungskonzepten basieren.

Microsoft Windows/286 läuft auf den meisten Computern mit dem Intel-Prozessor 80286 als CPU. Windows/386 ist auf dem Compaq-Deskpro-386 und kompatiblen Computern sowie auf dem IBM-PS/2-Modell 80 lauffähig. Windows/286 erfordert eine physikalische Hauptspeicher-Kapazität von 512 Kbyte, eine Festplatte und einen EGA- oder VGA-Bildschirm-Adapter (oder entsprechende Adapter). Windows/386 setzt 2-Mbyte-Hauptspeicher, eine Festplatte und einen Grafikadapter voraus.

Microsoft Windows/286, /386 und das Toolkit Version 2.1 sind in allen gängigen Diskettenformaten lieferbar. Anwender der jeweiligen Vorgänger-Versionen 2.03 können das Update auf die neuen Versionen 2.1 für 50 Mark (exkl. MwSt.) erhalten. Das Upgrade von Windows 2.03 auf Windows/386 2.1 kostet 399 Mark.

Neue Windows Versionen steigern die Leistung von Excel

Die neuen Versionen 2.1 der Betriebssystem-Erweiterung Microsoft Windows/286 und Windows/386 machen das Tabellenkalkulations-Programm Microsoft Excel jetzt noch schneller.

Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, betont: »Microsoft Excel ist darauf ausgelegt, die Möglichkeiten und Leistungsfähigkeit der neuen Hardware-Generation zu nutzen. Da die Anwender zunehmend auf Computer mit 80286- oder 80386-Prozessor-Basis umsteigen und in der Regel große komplexe Modelle erstellen, können sie in hohem Maße von den Verbesserungen profitieren, die die neuen Windows-Versionen bringen.«

Zusätzlich zu der erheblichen Leistungssteigerung auf PCs mit 80286-Prozessor-Basis bieten Microsoft Windows/286 und Windows/386 einen um 45 Kbyte größeren adressierbaren Hauptspeicher, falls mehr als 1 Mbyte Hauptspeicher zur Verfügung steht. Das in die beiden

Mitteilungen .... Mitteilungen .... Mitteilungen

neuen Windows-Versionen integrierte Installationsprogramm vereinfacht das Einrichten und die Anpassung von MS Windows. Die folgenden Benchmarks wurden mit Microsoft Excel 2.01 und MS Windows 2.03 gegen MS Excel 2.01 mit MS Windows/286 durchgeführt. Die meisten Tests liefen auf einem IBM-PS/2-Modell 60. Die Netzwerk-Benchmarks sind auf einem Compaq-Deskpro-386/20 durchgeführt worden. Es ließen sich dabei erhebliche Leistungssteigerungen in allen Bereichen von Microsoft Excel feststellen. Vergleichbare Leistungsverbesserungen sind zu erwarten, wenn anstatt Windows/286 die neue Version von Windows/386 eingesetzt wird.

Netzwerk: Bei Betrieb von Windows und Excel auf einem Netzwerk stieg die Leistung um 87 Prozent.

Druck: Die Druckgeschwindigkeit von Excel stieg um 22 Prozent. Eingesetzt wurde ein HP-LaserJet, Serie II.

Makros: Die Ausführungsgeschwindigkeit von Excel Makros war um 48 Prozent höher.

Menüs: Der Excel Menü-Aufruf war um 42 Prozent schneller.

Microsoft Excel ist das erste Zahlenmanagement-Programm für MS-DOS, bei dem die umfangreichen grafischen Möglichkeiten von Microsoft Windows zum Einsatz kommen. Das Programm läuft auf IBM PS/2-, Compaq-Deskpro 386- und AT-Kompatiblen Computern. Auf IBM-PC/XT und kompatiblen Computern läuft Microsoft Excel mit guter Performance, wenn diese PCs mit Microsoft MACH20 und einer Speichererweiterung ausgerüstet sind.

Der OS/2 LAN Manager bietet Netzwerkunterstützung für Windows-Anwendungen

Die neuen und jetzt verfügbaren Versionen Windows/286 2.1 und Windows/386 2.1 der grafischen Betriebssystem-Erweiterung Microsoft Windows unterstreichen eindrucksvoll die Bedeutung von Windows-Workstations in PC-Netzwerken. Da heutige Netzwerke aus einer Kombination von MS-DOS und MS OS/2 basierenden Workstations bestehen, bieten Microsoft Windows und der Microsoft Presentation Manager eine einheitliche Bedienungsoberfläche in beiden Umgebungen. Mit dem Microsoft OS/2 LAN Manager, der Windows-Workstations unterstützt, können Windows-Anwendungen alle Vorteile der Netzwerkleistungen und der »Application Programming Interfaces« (APIs) nutzen, die ein integraler Bestandteil der MS OS/2 LAN Manager-Technik sind.

Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, unterstreicht die Bedeutung der Anschlußfähigkeit, »die ein immer wichtigerer Faktor für die Produktivität von Arbeitsgruppen ist. Der MS OS/2 LAN Manager ermöglicht den Windows-Anwendern den Anschluß an ein leistungsfähiges lokales Netzwerk auf MS OS/2-Basis, das die Netzwerkleistungen für die gesamte Arbeitsgruppe bereitstellt.«

Mit dem MS OS/2 LAN Manager bietet Microsoft eine Systemsoftware, die einer Arbeitsgruppe innerhalb eines Netzwerkes Funktionen wie Datei-, Druck-, Ressourcen-, Sicherheits- und Netzwerk-Management einräumt. Der LAN Manager setzt MS OS/2 auf dem Server voraus und unterstützt MS Windows-, MS-DOS- und MS OS/2-Workstations. Wie Windows und MS OS/2 Presentation Manager entspricht das Bedienungsinterface des LAN Managers der »Common User Access Specification« (CUA) innerhalb der IBM-»Systems Application Architecture« (SAA). Der MS OS/2 LAN Manager erweitert die MS-DOS und MS OS/2 Programm-Schnittstellen quer über das gesamte Netzwerk und bietet umfangreiche netzwerkspezifische APIs.

Anwendungen, die für Microsoft Windows geschrieben sind, arbeiten in vollem Maße auch in MS OS/2 LAN Manager-Netzwerken. Sie ermöglichen über den MS OS/2 LAN Manager den Zugriff auf Massenspeicher-Laufwerke, Drucker und andere Einheiten innerhalb des Netzwerkes. Da Windows-Anwendungen die APIs des MS OS/2 LAN Managers nutzen, können sie auch auf MS OS/2 LAN Manager basierende Anwendungen wie den SQL Server zugreifen. Besonders wichtig für die Interprozeß-Kommunikation sind in diesem Zusammenhang die »Named Pipes« und »Mail Slots«. Diese Mechanismen sorgen dafür, daß der Verarbeitungsbedarf auf die verschiedenen Rechner im Netz verteilt wird, so daß die Kapazität der PCs voll genutzt werden kann.

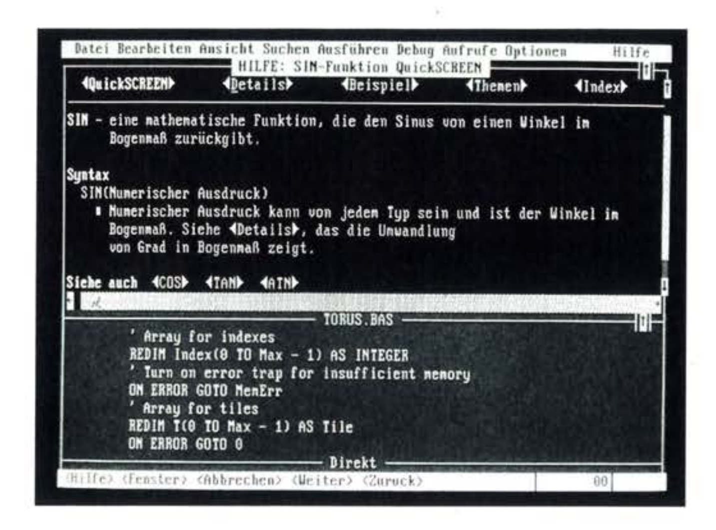
MS OS/2 LAN Manager jetzt auch für große Netzwerke und mehrere Benutzer

Die von Microsoft angekündigte Erweiterung des MS OS/2 LAN Managers wird inzwischen ausgeliefert. Diese Erweiterung macht es möglich, daß mehrere Anwender gleichzeitig mit unterschiedlichen Anwendungen in einem LAN Manager-Netzwerk arbeiten können. Microsoft stellt diese intelligente Upgrade-Version seinen MS OS/2 LAN Manager OEM-Kunden kostenlos zur Verfügung.

Wesentliches Merkmal der vorgenommenen Erweiterung ist die Erhöhung der »File Handles« im MS OS/2 LAN Manager von bislang 255 auf nunmehr 64.000. Mit dieser Vergrößerung der Anzahl verfügbarer Dateien wird praktisch eine unbegrenzte Zahl von Anwendern unterstützt, die eine entsprechend große Anzahl von Programmen gleichzeitig benutzen können.

Wieviele Benutzer gleichzeitig auf Anwendungen zugreifen können, hängt bei der neuen Version des MS OS/2 LAN Manager von der Art der Anwendung ab, da verschiedene Programme jeweils eine unterschiedliche Anzahl von »File Handles« benutzen. So erhöht beispielsweise die erweiterte LAN Manager-Version bei Programmen wie Lotus 1-2-3, Microsoft Excel oder Microsoft Word die Zahl der Benutzer von rund 40 auf 240.

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen



OnLine-Ratgeber im Microsoft QuickBASIC 4.5 hilft Programmier-Anfängern

Mit QuickBASIC 4.5 werden dank grundlegender Verbesserungen auch Programmier-Anfänger in die Lage versetzt, schnell die wichtigsten Aufgaben bei der Programmierung zu meistern. Microsoft begann in diesen Tagen mit der Auslieferung der deutschen Version.

Der Microsoft QuickBASIC-4.5-Ratgeber, ein elektronischer »Lehrer«, sowie verkürzte Menüs und Lernlektionen am Bildschirm und auf Papier führen zu einer schnell erreichbaren Programmier-Produktivität. Der QuickBASIC-Ratgeber ist ein revolutionäres Bildschirmlexikon, das unter Einsatz der Hypertext-Technologie dem Anwender eine Fülle von Informationen per Tastendruck zur Verfügung stellt. Darüber hinaus ermöglichen die einfach gestalteten Menüs dem Programmieranfänger, sich schnell in der Microsoft QuickBASIC-Programmierumgebung zurechtzufinden und produktiv zu werden. Die QuickBASIC-Expressfunktion unterrichtet den Anfänger über alle Punkte, die er wissen muß, um sich innerhalb kürzester Zeit in QuickBASIC 4.5 zu orientieren. Außerdem kann der Programmierneuling ausführliche gedruckte Tutorials heranziehen, um das Wesentliche der Programmierung innerhalb der Microsoft QuickBASIC-Umgebung zu lernen.

Neben der innovativen Hilfstechnik in Form des QuickBASIC-Ratgebers bietet Microsoft QuickBASIC 4.5 dieselben leistungsfähigen Funktionen wie Microsoft QuickBASIC 4.0, einschließlich eines »intelligenten« Syntax-Prüfeditors, einer schnellen Compilierung mit Geschwindigkeiten von bis zu 150.000 Zeilen/Minute sowie einen verbesserten Source-Level-Debugger mit einer neuen schnellen Überwachungsfunktion.

»Microsoft QuickBASIC 4.5 geht ganz neue Wege, indem es den Anfänger sofort produktiv werden läßt, sobald er das Produkt ausgepackt hat«, so Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, zur Vorstellung des neuen Produkts, »Schon die Version 4.0 brachte einen erheblichen Fortschritt in der Technologie der Programmiersprachen. Mit den fortschrittlichen Lernwerkzeugen fügt Microsoft nun in der QuickBASIC-Version 4.5 einen weiteren Teil hinzu, der es möglich macht, in kürzester Zeit Programme zu schreiben.«

Microsoft QuickBASIC 4.5 ist vollständig kompatibel mit dem Microsoft BASIC Compiler 6.0. BASIC 6.0 ist eine komplette professionelle Entwicklungsumgebung. Sie bietet eine Reihe zusätzlicher Funktionen und Möglichkeiten, wie MS OS/2-Support, verbessertes Debugging mit Microsoft CodeView, den Microsoft Editor sowie Optimierungssteuerungen, so daß der professionelle Programmierer sein Endprodukt hinsichtlich Programmgröße und Geschwindigkeit maßschneidern kann.

Der QuickBASIC-Ratgeber gewährleistet erhöhte Produktivität

Wesentliches Merkmal des QuickBASIC-Ratgebers ist die kontextbezogene Online-Hilfe am Bildschirm. Mit ihr kann der Programmier-Anfänger die grundlegenden Probleme bei der Erstellung von Programmen meistern. Der QuickBASIC-Ratgeber ist schneller und einfacher zu handhaben als ein Handbuch. Dank der modernen Technik dieser Bildschirmhilfe sind gedruckte Handbücher und Schnellübersichten nahezu überflüssig. Ein einziger Tastendruck oder ein entsprechendes Klicken mit der Maus bringt sofort kontextbezogene Erläuterungen zu Programm-Anweisungen, Fehlermeldungen (Syntax und Laufzeitfehler), Dialogfenstern, Menüinhalten und natürlich auch über das Hilfesystem selber auf den Bildschirm.

Basierend auf der Hypertext-Technologie ermöglicht die Hilfefunktion einen extrem schnellen Zugriff auf umfangreiche Einträge mit Querverweisen. Sie enthält das komplette Befehlsverzeichnis plus zahlreiche Beispielprogramme, die direkt in ein eigenes Programm kopierbar sind.

Durch die Positionierung des Cursors auf einen beliebigen Punkt des Bildschirms, zum Beispiel auf eine Programmanweisung, eine Fehlermeldung, ein Dialogfenster oder einen Menüpunkt, erhält der Anwender nach Drücken der Maustaste sofort eine vollständige Erklärung des angeklickten Punktes. Außerdem kann er per Querverweis direkt in andere themenbezogene Begriffe springen und Erläuterungen zu ihm unbekannten Begriffen oder zu der Hilfefunktion aufrufen. Für Benutzer einer Maus ist der Zugriff auf diese Informationen so einfach wie das Positionieren des Cursors und das Klicken der Maustaste.

Möchte ein Programmierer wissen, wie ein Variablentyp oder eine Datenstruktur im jeweiligen Programm eingesetzt wurde, so gibt der QuickBASIC-Ratgeber schnell Auskunft darüber, wofür der jeweilige Variablentyp oder die Datenstruktur steht, an welcher Stelle sie eingesetzt sind und in welcher Beziehung sie zu anderen Variablentypen oder Datenstrukturen innerhalb des Programms stehen. Die Hilfefunktion bietet außerdem Informationen über die Grenzen der Programmierumgebung sowie schnellen Zugriff auf ASCII-Tabellen.

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen

Einfache Menüs bieten zusätzliche Hilfe

Ein weiteres Merkmal, das Anfängern hilft, sich beim Programmieren schnell zurechtzufinden, ist die »Easy Menus Option«. Sie vereinfacht das Bediener-Interface durch die Reduzierung der Auswahlmöglichkeiten in den Menüs und Dialogfenstern. Bei der Installation von QuickBASIC werden sofort automatisch kurze, einfache Menüs eingerichtet. Diese Option unterstützt den Programmier-Anfänger dadurch, daß er am Anfang nur eine geringe Anzahl von Kommandos kennen muß, um zu programmieren. Sobald die Grundlagen »sitzen«, kann der Programmierer sich die vollständigen Menüs darstellen lassen.

Die Tutorials

Microsoft QuickBASIC 4.5 wird sowohl mit bildschirmgestützten als auch mit gedruckten Lektionen geliefert, die eine weitere Hilfe für den Einsteiger sind. Microsoft QuickBASIC Express ist eine schnelle und anwendungsfreundliche Bildschirm-Lernhilfe, die dem Anfänger alles erläutert, was er braucht, um sich in der Microsoft QuickBASIC Umgebung zurechtzufinden. Die gedruckten Lektionen sind so angelegt, daß der Anwender Schritt für Schritt durch den Prozeß zur Erstellung einer elektronischen Kartei geführt wird.

Integrierter Editor und Debugger bringen Programme schneller zum Laufen

Wie die Vorgängerversion von Microsoft QuickBASIC bietet die Version 4.5 das außergewöhnliche »Instant Environment«, das aus einem integrierten Editor und einem Source-Level-Debugger besteht. Mit dieser integrierten Programmierumgebung können Anwender auf einfache Weise zwischen dem Programmschreiben und der Fehlersuche wechseln, ohne die Programmierumgebung zu verlassen. Das einzigartige Feature »Edit-and-Continue« bietet die Möglichkeit, daß der Programmierer ein ablaufendes Programm unterbrechen, Änderungen einfügen und dann das Programm sofort weiterlaufen lassen kann. Der integrierte Debugger neuester Technik läßt durch Tastendruck die Fehlersuche zu jeder Zeit der Programmentwicklung zu. Mit der neuen Funktion »Anzeige aktueller Werte« kann der Programmierer sofort den Wert jeder Variablen und jedes Ausdrucks feststellen.

Hardware-Voraussetzungen und Verfügbarkeit

Die Mindestsystemkonfiguration für den Einsatz von Microsoft QuickBASIC 4.5 ist ein IBM-PC oder kompatibler Computer mit 384 Kbyte Hauptspeicher, MS-DOS 2.1 Betriebssystem (oder höhere Version) und zwei 360 Kbyte 5 1/4"-Laufwerke bzw. ein 720 Kbyte 3 1/2"-Laufwerk.

Microsoft QuickBASIC 4.5 unterstützt die Microsoft Maus sowie CGA-, VGA-, EGA- und Herkules-Grafikkarten. Das Produkt ist ab sofort erhältlich.

Daten und Fakten zu Microsoft QuickBASIC 4.5

Microsoft QuickBASIC 4.0 war bei seiner Vorstellung der Protagonist einer neuen, fortschrittlichen Programmiersprachen-Technologie. Er vereinte die am weitesten entwickelte Interpreter-Technologie mit der am meisten benutzten Programmiersprache, Microsoft BASIC. Das Ergebnis war eine integrierte Programmierumgebung, die schnelle Ergebnisse während des Entwicklungszyklus bietet.

Mit QuickBASIC 4.5 geht Microsoft jetzt den Weg des »Instant«-Konzepts noch einen Schritt weiter, und zwar in zwei wesentlichen Bereichen: Orientierungshilfe durch das Anwender-Interface sowie Lernhilfen sind nun Bestandteil des Paketes. Diese Lernhilfen umfassen innovative, bildschirmgestützte Hilfefunktionen und ein neues Computer Based Training (CBT).

Microsoft hat das Hauptaugenmerk bei QuickBASIC 4.5 auf die Produktivität gerichtet und bietet die fortschrittlichsten Hilfen für die am einfachsten zu benutzende Programmiersprachen-Umgebung an. Damit Software einfacher erlern- und anwendbar wird, ist es erforderlich, daß die bildschirmorientierten Hilfen ebenso schnell und einfach verfügbar sind. Eine innovative Online-Hilfetechnik, wie der Microsoft QuickBASIC-Ratgeber als Teil des Microsoft QuickBASIC 4.5 Paketes, bietet sowohl dem Anfänger wie auch dem erfahrenen Programmierer einen raschen Zugriff auf die Informationen, die er benötigt. Der Microsoft QuickBASIC-Ratgeber ist ein bildschirmgestütztes Nachschlage-/Hilfesystem, das auf der »Hypertext«-Technik basiert. Für bestimmte Programmierer kann der QuickBASIC-Ratgeber die Kluft schließen, die bislang Makro-Programmierer von Tabellenkalkulationen und Datenbanken davon abhielt, leistungsfähigere Programmierwerkzeuge, wie QuickBASIC, einzusetzen.

Der Microsoft QuickBASIC-Ratgeber macht Kurzübersichten überflüssig. Man kann ihn nicht verlieren und er ist schneller und auch vollständiger als eine Kurzübersicht. Der QuickBASIC-Ratgeber ist darüber hinaus jederzeit mit einem Tastendruck oder einem Mausklick aktivierbar und gibt sofort Antwort. Er bietet ein vollständiges Nachschlageverzeichnis der Programmiersprache auf dem Bildschirm, das den Zugriff auf die indexierten Informationen sehr viel schneller erlaubt als gedruckte Kurzbeschreibungen.

Auch die Handbücher kann der Anwender nun beiseite legen. Ein Durchblättern von mehreren hundert Dokumentationsseiten, um eine bestimmte Detailinformation zu finden, ist nicht mehr erforderlich. Mit dem Microsoft QuickBASIC-Ratgeber stehen die kompletten Informationen eines Handbuchs sofort auf dem Bildschirm zur Verfügung. Darüber hinaus ist es sogar möglich, Beispielprogramme aus dem Ratgeber mit wenigen Tastenfolgen in den Editor zu kopieren und sofort auszutesten.

Die Technologie des Microsoft QuickBASIC-Ratgebers wird auch in zukünftigen Versionen anderer Microsoft-Produkte, wie CodeView, Debugger und im Microsoft Editor

Mitteilungen .... Mitteilungen .... Mitteilungen

eingesetzt. Da das Ratgeber-Format in allen Programmen gleich ist, kann ein Programmierer ein Programm in einer Microsoft-Sprache schreiben und auf Hilfedateien zurückgreifen, die in anderen Microsoft-Sprachen und -Utilities enthalten sind.

Eine weitere wichtige Verbesserung im Microsoft QuickBASIC-4.5-Paket ist »QuickBASIC Express«. Dieses computergestützte Trainingssystem wurde von jenem Team entwickelt, das auch für das preisgekrönte »Microsoft Learning DOS Training Product« verantwortlich zeichnete. Microsoft QuickBASIC 4.5 bietet ein interaktives Lernprogramm unter Nutzung der neuesten CBT-Technologie. Es ist so angelegt, daß ein Benutzer schon nach zehn Minuten in der Microsoft QuickBASIC-4.5-Umgebung produktiv werden kann. Ein zusätzliches gedrucktes Trainingsbuch führt Schritt für Schritt durch die Entwicklung eines elektronischen Karteiprogramms.

Microsoft QuickBASIC 4.0 ist ein leistungsfähiges System zur Programmentwicklung mit einer großen Zahl von Menü-Kommandos. Bei Gesprächen mit Microsoft QuickBASIC-Anwendern stellte sich heraus, daß die Komplexität des Bediener-Interfaces eines der wesentlichen Hindernisse war, um mit Microsoft-QuickBASIC 4.0 produktiv zu arbeiten. Es mußte also die Bedienungsoberfläche an den Programmieranfänger angepaßt werden. Daraus durften aber keine Leistungsabstriche für den erfahrenen Benutzer folgen. Die Lösung des Problems ist »Easy Menus«, ein »doppelgleisiges« Bediener-Interface: eines für den Programmier-Neuling und ein anderes für den erfahrenen Programmierer. Microsoft hat eine Reihe von verschiedenen Bedienungsoberflächen auf ihre Verwendbarkeit hin überprüft. »Easy Menus«, so wie es in QuickBASIC 4.5 implementiert wurde, ist das Ergebnis eines Entwicklungsprozesses, bei dem alles berücksichtigt wurde, was Programmier-Anfänger in die Lage versetzt, so schnell wie möglich produktiv zu werden.

Für den Programmierneuling werden die Befehle bereitgestellt, die notwendig sind, um zu entwickeln, Fehler zu suchen und einen Programmlauf durchzuführen. Wenn der Anwender die volle Leistung des Microsoft QuickBASIC-Interface benötigt, so wählt er »Vollständiges Menü« aus dem Optionen-Menü aus.

Natürlich bietet QuickBASIC 4.5 kontextbezogene Hilfen bei jeder Menü-Option in der Statuszeile am unteren Bildschirmrand. Möchte der Anwender weitergehende Erklärungen zu »Speichern unter...«, so wählt er einfach »Speichern unter...« und drückt die [F1]-Taste. Diese Taste ist in allen Programmen die Hilfe-Taste.

Im Modus »Easy Menus« wurde nicht nur die Menüstruktur vereinfacht, sondern auch die Dialogfenster. So hat beispielsweise im Modus »Easy Menus« das »EXE Datei erstellen«-Fenster nur eine Option: Erzeugung eines Stand-Alone-EXE-Files. Bei vorheriger Auswahl von »Vollständiges Menü« hat das »EXE Datei erstellen...«-Dialogfenster dagegen zwei Optionen: Stand-Alone und BRUN. »Easy Menus« ist ein Element von vielen, die darauf ausgelegt sind, dem Anfänger die Programmierung mit QuickBasic so einfach und schnell wie möglich zu machen.

Benutzer einer Maus können jetzt die Funktion der rechten Maustaste mit Hilfe eines Befehls aus dem Optionen-Menü so einstellen, daß eine kontextbezogene Hilfe auf dem Bildschirm erscheint. Außerdem kann der Anwender Pfade aus der Programmierumgebung heraus setzen, um die Dateien nach dem Typ zu organisieren und in verschiedenen Verzeichnissen zu speichern.

Eine neue Debugging-Funktion in Microsoft QuickBASIC ist der Befehl »Anzeige aktueller Werte« zur schnellen Feststellung des Wertes einer Variablen oder eines Ausdrucks.

Ebenfalls eine Novität ist die variable Hilfefunktion, die Informationen liefert, wenn der Cursor auf das entsprechende Symbol positioniert und die [F1]-Taste bzw. die Maustaste gedrückt wird. Unter Nutzung der Symboltabellen, die für diese Programmierumgebung erstellt wurden, liefert die variable Hilfefunktion Informationen über den Datentyp, über Prozeduren, Variable und Datenstrukturen. Diese Möglichkeit erspart es dem Anwender, den kompletten Source-Code eines Programms durchzuforschen, um die notwendigen Informationen über die Symbole zu erhalten, die in dem Programm genutzt werden.

Drei neue Anweisungen bieten die Möglichkeit, vom Anwender definierte Ereignisse (events) abzufangen. Diese Anweisungen sind besonders in industriellen Anwendungen der Steuer- und Meßtechnik nützlich, wo es um Interrupt-Verarbeitung geht. ON UEVENT definiert eine Ereignisverfolgung für ein anwenderdefiniertes Ereignis. UEVENT ON/OFF/STOP initialisiert, deinitialisiert oder setzt das Trapping eines anwenderdefinierten Ereignisses aus. SLEEP setzt die Ausführung eines BASIC-Programms aus.

Bislang führte ein Fehler in einem Programm-Modul ohne Fehlerbehandlungsroutine, zur sofortigen Beendigung des Programms. Anders beim QuickBASIC 4.5: Wenn ein Fehler in einem Modul ohne Fehlerroutine auftritt, verfolgt QuickBASIC die Prozedur-Aufrufe rückwärts, bis eine aktive Fehler-Behandlungsroutine gefunden wurde oder das Hauptmodul des Programms erreicht ist.

Wie die vorhergehende Version von Microsoft QuickBASIC, so bietet auch die Version 4.5 das revolutionäre »Instant Environment«, bestehend aus einem integrierten Editor, einem Compiler und einem Source-Level-Debugger. Diese integrierte Programmumgebung ermöglicht es den Anwendern, auf einfache Weise zwischen Editieren, Compilieren und Debuggen hin- und herzuwechseln, ohne die Programmierumgebung verlassen zu müssen. Dies Möglichkeiten in Microsoft QuickBASIC 4.5 umfassen:

- Kompilierung von 150.000 Zeilen/Minute (gemessen auf einem IBM-PC/AT bei 8 MHz Taktfrequenz).
- Eine »Edit-and-Continue«-Funktion, die es möglich macht, den Programmlauf zu unterbrechen, Verände-

Mitteilungen .... Mitteilungen .... Mitteilungen

rungen im Programm vorzunehmen und dann den Programmlauf an genau der Stelle fortzusetzen, wo er unterbrochen wurde. Alle Datenwerte bleiben dabei unangetastet.

- Eine Syntaxüberprüfung, die dem Programmierer hilft, syntaxgerechte Programme zu schreiben, indem Syntax-Fehler sofort bei der Eingabe festgestellt werden.
- Ein Ausführungsmodus »Direkt« erlaubt dem Anwender das Öffnen eines Fensters, wobei eine einzige Programmzeile getestet wird, ohne das gesamte Programm ablaufen lassen zu müssen.
- Eine automatische Erzeugung von Stand-Alone oder mehrfach-modularen Programmen über Menü-Auswahl.
- Einen integrierten Debugger auf dem neuesten Stand der Technik, der mit einem einzigen Tastendruck zu jeder Zeit der Programmentwicklung die Fehlersuche möglich macht. Die wesentlichen Debugging-Funktionen, zusätzlich zu »Anzeige aktueller Werte« und variabler Hilfe sind:
  Haltepunkte, die bei jeder Anweisung gesetzt und wieder entfernt werden können, so daß der Anwender ein Programm an jedem beliebigen Punkt stoppen kann, um zu sehen, was passiert.
  Stoppbedingungen, die einen Programmlauf stoppen, wenn bestimmte Bedingungen zutreffen (TRUE).
  Ein *Debug-Fenster*, das die Werte von bis zu acht Ausdrücken gleichzeitig anzeigt, wobei die Wertänderungen im Zuge des Programmablaufs dargestellt werden.
  Einzelschritt-Ausführung und *Programm-Kontrolle* erlauben dem Anwender, jeden Schritt der Programmausführung zu sehen.

Wie Microsoft QuickBASIC 4.0, so unterstützt auch die Version 4.5 die modulare und strukturierte Programmierung, um mögliche zukünftige Änderungen des Programms zu vereinfachen. Schlüsselfunktionen sind dabei:

- Alle wichtigen strukturierten Programmier-Konstrukte, wie SELECT CASE-, Block-, IF-THEN-ELSE-END-, IF- und DO-Anweisungen.
- »Teil«-Fenster für Prozeduren, Funktionen oder Module, so daß der Anwender den Programm-Code in zwei Bereichen des Programms gleichzeitig betrachten kann.
- Ein integrierter »Code Outliner«, der sofort die Struktur eines Programms anzeigt.

Andere fortschrittliche Funktionen sind:

- Unterstützung von benutzerdefinierten Typen oder Records, TRUE-Funktionen, rekursive Prozeduren/Funktionen, Zeichenketten mit fixierter Länge sowie Arrays, die größer als 64 Kbyte sind.
- »Mixed-Language«-Programmierung - die es dem Anwender ermöglicht, Programm-Code einzubinden, der in anderen Microsoft-Sprachen geschrieben wurde.

Die Microsoft Programmer's Library ist jetzt da

Die Microsoft Programmer's Library ist ab sofort im Handel erhältlich. In dieser CD-ROM Text-Datenbank sind 48 Handbücher zu den Produkten MS OS/2, Microsoft Windows, MS-DOS, Microsoft C-Compiler, Microsoft BASIC, Microsoft Makro-Assembler, Microsoft Pascal sowie Microsoft FORTRAN und Microsoft Hardware gespeichert. Darüber hinaus stehen 1.200 Musterprogramme und andere technische Dokumentationen zur raschen Verfügung.

Auf alle Informationen kann der Anwender direkt zugreifen. Er muß dabei das laufende Programm nicht verlassen, um Informationen aus der Programmer's Library auf seinen Bildschirm zu holen, sondern kann diese mit Hilfe eines Editors oder eines Textverarbeitungsprogramms direkt bearbeiten. Von dieser schnellen und einfachen Online-Zugriffsmöglichkeit profitieren nicht nur professionelle Software-Entwickler, sondern auch Autoren technischer Beiträge oder Spezialisten des Produkt-Supports. Zur Unterstützung des raschen Zugriffs wurde Microsofts Programmer's Library mit einer hochentwickelten Suchfunktion ausgestattet, die der Komplexität des Inhalts gerecht wird und vor allem den Erwartungen der Anwender an ein elektronisches Nachschlagewerk entspricht. Microsoft Programmer's Library ist ab sofort für ca. 680 Mark ohne MwSt. (unverb. Preisempfehlung) im Handel erhältlich.

Weltweit erster Volumenmodellierer für OS/2

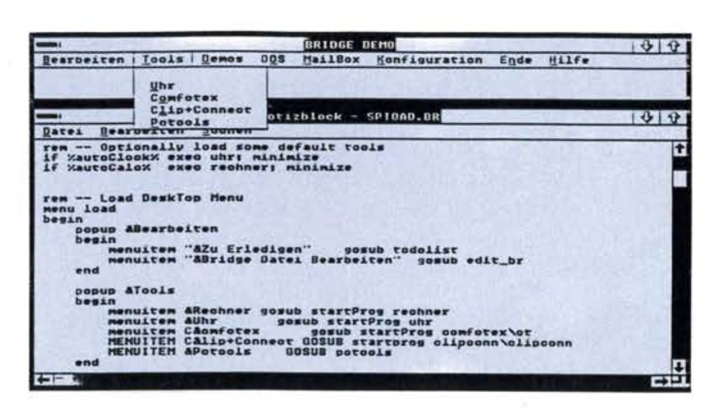
Mit Desikon stellt das Hamburger Unternehmen rotring euroCAD ein von Grund auf neu entwickeltes CAD/CAM-System vor, das innovative Technologie für mechanische Konstruktion und Fertigung mittelständischer Produktionsbetriebe bietet.

Funktional liegt ein integriertes Draht-/Flächen-/Volumenmodell vor, dessen Daten für Konstruktion und Fertigung von vornherein nur einmal erzeugt werden. Somit entfallen doppelte Datenhaltung sowie unnötige Konvertierungen innerhalb des Systems. Über das Festlegen der Geometrien hinaus beinhaltet der Leistungsumfang einen integrierten BASIC-Interpreter. Er ermöglicht sowohl die grafische Ausgabe von Berechnungsprogrammen als auch ein »Mitschreiben« grafisch-interaktiver Tätigkeiten für Variantenkonstruktionen. Als »eingebauter« Taschenrechner oder zum Aufbau von Bibliotheken mit variablen Teilen läßt er sich ebenfalls einsetzen. Eine Applikationsschnittstelle sichert die redundanzfreie Kommunikation.

Für den Fertigungsbereich bereitet Desikon die Daten für NC-Programmiersysteme fertigungsspezifisch auf und hält sie in einem neutralen APT-Format bereit.

Die Software wurde unter OS/2 entwickelt und nutzt alle Möglichkeiten dieses neuen Betriebssystems.

Mitteilungen .... Mitteilungen .... Mitteilungen



Eine Brücke von DOS zu MS-Windows

Mit ComfoBridge bietet die Software Products International (Deutschland) GmbH (SPI) eine komfortable Verbindung von DOS- und Windows-Anwendungen. Im Einzelnen bietet ComfoBridge, als Neuheit für Microsoft Windows-Anwender, eine strukturierte Batch-Programmierung mit Schnittstellen zu MS-DOS und MS Windows, Interpreter und Run-Time-Funktion und ein Mailboxsystem zum Datenaustausch zwischen Anwendungen.

Alle DOS-Programme, auch individuelle Anwendungen, können unabhängig von der verwendeten Programmiersprache mit einer Windows-Oberfläche versehen werden. In Verbindung mit Windows/386 ist beim Ablauf Multitasking möglich, d.h. während der Anwender sich in Windows bewegt läuft seine DOS-Anwendung simultan im Hintergrund. Werden z.B. in einer Windows-Schirmmaske Daten der Finanzbuchhaltung erfaßt, erfolgt gleichzeitig in der DOS-Anwendung die Verarbeitung. Komplexe Anwendungen können so nachträglich mit einer komfortablen Benutzeroberfläche versehen werden. ComfoBridge bietet einen Ausweg aus dem bislang noch bei Windows bestehenden Manko der mangelhaften Zahl verfügbarer Anwendungen. Die gesamte Palette der für DOS erhältlichen Programme kann auch unter der grafischen Benutzeroberfläche genutzt werden. Selbst komplexe Anwendungen können mit der grafischen Benutzeroberfläche versehen und mit der Maus anwendungsorientiert bedient werden.

Mit der Batch-Programmierung von ComfoBridge können komplexe Lösungen erstellt werden. ComfoBridge als Integrator der DOS- und Windows-Welt erlaubt dabei den Zugriff auf Programme beider Welten. Die Verknüpfung einer Datenbank unter DOS mit einer Textverarbeitung unter Windows ist problemlos möglich. Die Syntax der ComfoBridge-Programmierung geht über den in DOS verfügbaren Umfang hinaus und bietet spezifische Windows-Elemente wie »Move« und »Restore«. Erste Praxiserfahrungen haben gezeigt, daß mit ComfoBridge ein Programmierumfang von 270.000 Zeilen verwaltet werden kann. Der Vergleich mit einer komplexen Software zeigt, daß nicht die Software sondern nur die verfügbare Hardware den Umfang von ComfoBridge Anwendungen einschränkt. Die Befehlsstruktur von ComfoBridge ist im Vergleich zur Programmierung mit einer Hochsprache wesentlich leichter zu erlernen, so daß schneller Ergebnisse erzielt werden können. Mit der »Browser«-Funktion steht dem Entwickler dabei ein Entwicklungs- und Testwerkzeug zur Verfügung.

Die Möglichkeit des Electronic Mail von ComfoBridge erlaubt bei der Programmierung von Anwendungen im Netz die Verknüpfung von Arbeitsplätzen. Die Eingabe an einem Arbeitsplatz kann z.B. eine Berechnung in einem anderen System auslösen. Die Erstellung eines Lieferscheins führt beispielsweise zur Reduzierung des Lagerbestandes am Arbeitsplatz eines zweiten Mitarbeiters.

EXE-, BAT- und COM-Dateien, z.B. Autoexec-Dateien, können mit ComfoBridge Befehlsabläufe auch unter MS-Windows automatisieren. War die Möglichkeit, Programmschritte in einer Datei zusammenzufassen bislang auf die DOS-Ebene beschränkt, können nun auch MS-Windows und Anwendungen unter Windows mit einbezogen werden. Der Anwender kann nach dem Start seines Systems direkt in eine bestimmte Datenerfassungsmaske gelangen. Das Starten einer Anwendung, die Wahl einer Datei und die Auswahl der Maske kann zusammengefaßt werden.

ComfoBridge bietet dem professionellen Entwickler umfangreiche Möglichkeiten zur Entwicklung neuer Anwendungen unter MS-Windows. Gleichzeitig dient das Programm dabei als Run-Time-Modul für die Anwendung. Nutzer der grafischen Benutzeroberfläche MS-Windows können verfügbare DOS-Anwendungen in ihre Windows-Oberfläche einbinden, ohne die Anwendung selbst ändern zu müssen. Dabei reichen die Möglichkeiten bis hin zu vollkommen individuellen Menüs.

MagicCV löst das CodeView-Speicherproblem

Der CodeView-Debugger ist ein sehr wichtiges Werkzeug in der Microsoft-Programmierumgebung, da er in komfortabler Form dem Entwickler die Möglichkeit gibt, seine Programme auf der Quellebene seiner verwendeten Hochsprache zu testen.

Ein Nachteil von CodeView ist der große Programmumfang; CodeView benötigt 250 Kbyte Speicher. Wenn man berücksichtigt, daß noch Speicher für Daten benötigt wird und eventuell speicherresidente Programme geladen sind, ist es in der Praxis oft nicht möglich, Programme mit mehr als 250 Kbyte zu testen. Als Folge hiervon kann der CodeView-Debugger bei größeren Programmen nicht mehr zum Testen eingesetzt werden und es kommen deshalb einfachere Werkzeuge zum Einsatz, die eine zügige Software-Entwicklung jedoch behindern.

Am günstigsten wäre es, wenn der Systementwickler auch bei geladenem CodeView-Debugger genausoviel Speicher zur Verfügung hätte, wie ohne ihn. Die Voraussetzung hierfür bietet auf der Hardwareseite der 80386-Prozessor mit seinem Konzept der virtuellen Maschine. Die notwendige Software-Unterstützung leistet das Programm Magic-

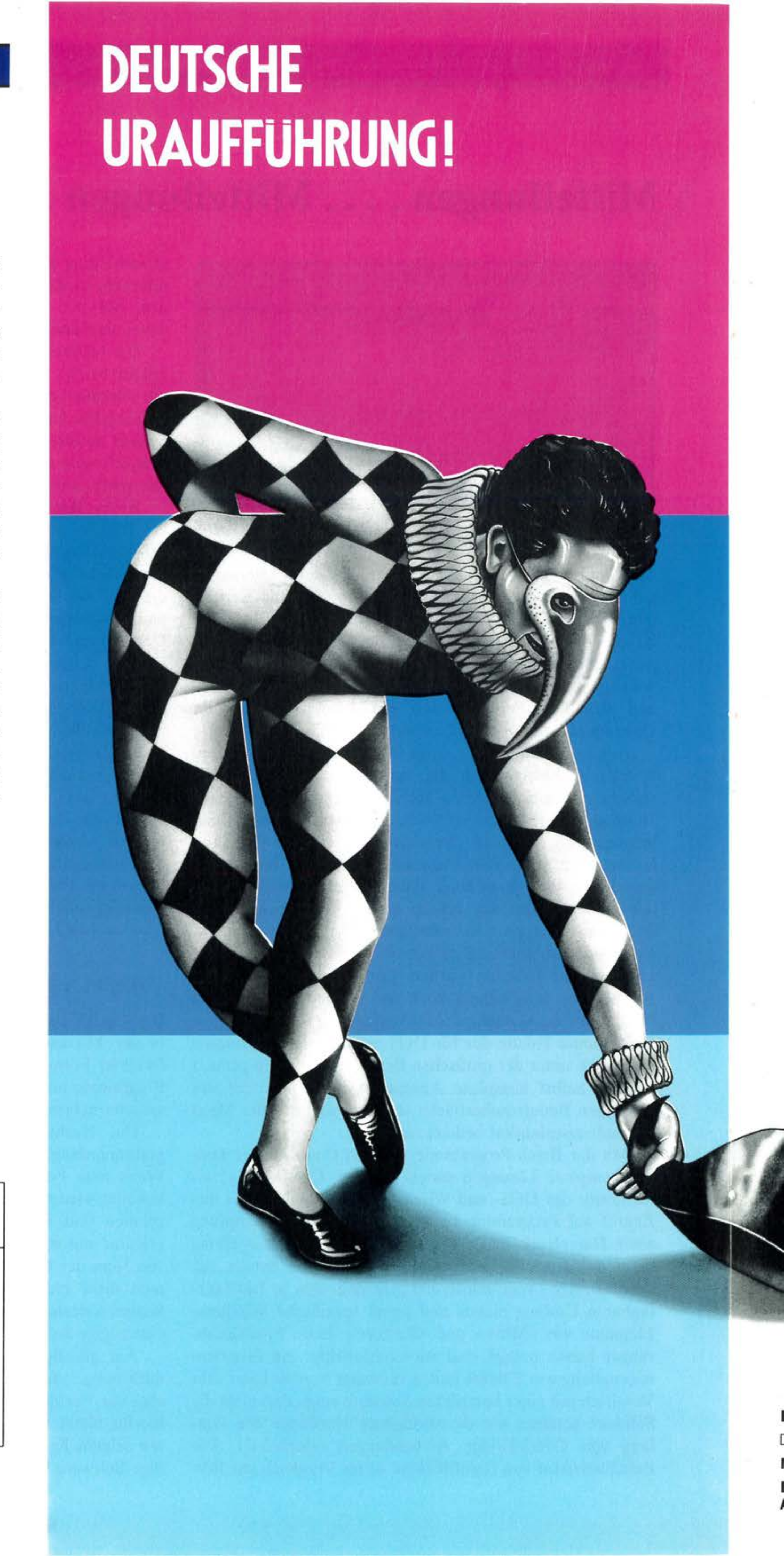
CV, welches zwei MS-DOS-Tasks einrichtet und das zu testende Programm sowie den CodeView-Debugger in die beiden Tasks lädt. Für die zweite Task benötigt MagicCV lediglich zwischen 4 und 8 Kbyte des Speichers unterhalb der 1-Mbyte-Grenze.

MagicCV ist ein eigenständiger Betriebssystemkern und arbeitet auf PC/AT-kompatiblen 80386-Rechnern mit mindestens 384 Kbyte Extended-Memory. Das Programm kann nicht zusammen mit anderen 80386-Programmen eingesetzt werden, die im Protected Mode arbeiten.

Mit Hilfe von MagicCV ist es nun möglich, auch bei größeren Projekten das enorme Rationalisierungspotential des CodeView-Debuggers zu nutzen. Die daraus resultierende Entwicklungszeitersparnis steht in keinem Verhältnis zum Anschaffungspreis von DM 685 (inkl. MwSt., Verpackung und Versand). MagicCV ist erhältlich bei der Firma PEM in Stuttgart-Möhringen.

Inserentenverzeichnis

BKS-Software	57
Computer 2000	12/13
Kickstein Software	37
Markt & Technik	Beilage
Markt & Technik Buchverlag	44/45, 112
Microsoft	56/57, 90/91
Niemeier	59
te-wi Verlag	2, 111
Zoschke	37



Ein schnelles, komfortables und umfangreiches Pascal von Microsoft:

Microsoft kündigt QuickPascal an

Am 26. April hat Microsoft in Moskau mit der weltweiten Vorstellung von Microsoft Quick-Pascal den Pascal-Massenmarkt betreten. QuickPascal ist ein Produkt, das durch Geschwindigkeit, umfangreiche Fähigkeiten und die Unterstützung erweiterter Programmierkonstruktionen neue Maßstäbe setzt.

QuickPascal ist schneller als jedes Pascal-Produkt im Personalcomputer-Markt. Es ist kompatibel zu Borlands Turbo Pascal und gleichzeitig preiswerter. Es verfügt über eine elegante Entwicklungsumgebung mit überlappenden Editier- und Debug-Fenstern, die bisher noch bei keinem Compiler zu finden waren. Es bietet sofortige Online-Dokumentation, die nur in der Microsoft Quick-Familie enthalten ist. Darüber hinaus wird damit gegenüber bisherigen Pascal-Produkten ein gewaltiger Sprung nach vorn getan, da QuickPascal die erste Pascal-Implementation auf PCs ist, die die objetorientierte Programmierung unterstützt.

»Die objektorientierte Programmierung wird der wichtigste Trend in der Programmierung in den 90er Jahren sein, genauso wie die strukturierte Programmierung selbst die wichtigste Entwicklung der 70er Jahre war«, meint dazu Christian Wedell, Geschäftsführer der Microsoft GmbH, die für den sowjetischen Markt verantwortlich ist. »Wir betreten den Pascal-Massenmarkt mit einem technisch überlegenen Produkt und mit einem Auge auf die Zukunft. Man erwartet von Microsoft einfach ein so gutes Sprachenprodukt.«

QuickPascal wurde als ideales Werkzeug zum Erlernen traditioneller und objektorientierter Programmierung entwickelt. Pascal selbst wurde entwickelt, um korrekte Programmiertechniken und den strukturierten Programmentwurf zu lehren. Objekte sind fortgeschrittene, selbständige Sprachstrukturen, die zur Kommunikation miteinander Meldungen verwenden. Die objektorientierte Programmierung kann den Programmentwurf vereinfachen und sie unterstützt die Wiederverwendung von Code, besonders bei grafischen Anwendungen, da Fenster, Dialogfelder, Menüs und andere Dinge einer Grafikumgebung als Objekte, die miteinander kommunizieren, behandelt werden können.

QuickPascal enthält umfangreiche Programmierfähigkeiten

QuickPascal ist eine leistungsfähige Implementierung von Pascal; sein Compiler und sein Linker sind die schnellsten auf PCs für Pascal verfügbaren. QuickPascal ist Quellcode-kompatibel zu Turbo Pascal; Programmierer, die Turbo Pascal-Programme entwickelt haben, können sie also einfach durch Neukompilierung laufenlassen. QuickPascal enthält die Units CRT, DOS, Printer und System zur Beschleunigung der Programmentwicklung und Vereinfachung der Wartung; es ist der einzige Pascal-Compiler mit Unterstützung für die Erzeugung von 286-Code.

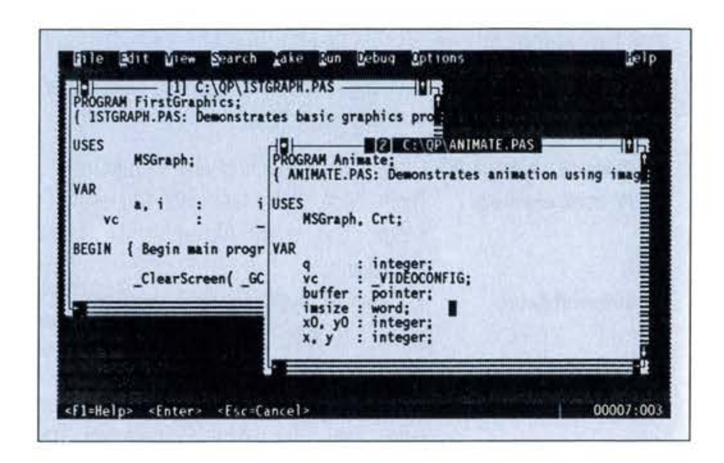


Bild 1: *QuickPascal verwaltet mehrere Programme nebeneinander in überlappenden Fenstern.*

Microsoft QuickPascal beinhaltet darüber hinaus eine ausgefeilte Grafikbibliothek, die Grundfunktionen und Routinen zum Zeichnen von Torten- und Balkendiagrammen sowie anderen Präsentationsgrafiken enthält.

Wie andere Mitglieder der weitverbreiteten Microsoft Quick-Sprachenfamilie integriert QuickPascal einen Editor, Debugger und Compiler und ermöglicht es Programmierern so, problemlos von einer zur anderen Funktion zu wechseln. Es verfügt auch über die Quick-Benutzerschnittstelle mit Drop-Down-Menüs und Dialogfeldern. Sofort verfügbare Online-Dokumentation wird durch den Quick-Pascal-Ratgeber geboten, ein Online-Referenzsystem auf der Basis von Hypertext. Darüber hinaus bietet die Quick-Pascal-Schnittstelle überlappende Editier- und Debug-Fenster.

Vor der Veröffentlichung von QuickPascal hat Microsoft nur ein High-End-Pascal-Produkt angeboten, den Microsoft Pascal-Compiler 4.0, der für professionelle Programmierer und die Entwicklung von großen Programmen unter MS-DOS oder MS OS/2 gedacht ist.

Objektorientierte Programmierung, der Programmierstil der Zukunft

Die objektorientierte Programmierung ist der nächste evolutionäre Schritt bei der strukturierten Programmierung. Anders als traditionelle Programmiersprachen, die Code und Daten separat halten, kapseln objektorientierte Sprachen sowohl Code und Daten zu einer einzigen Klasse zusammen.

Die objektorientierte Programmierung vereinfacht und beschleunigt die Programmierung, indem sie den Programmierer von den Details, wie ein Objekt arbeitet, isoliert. Die objektorientierte Programmierung vereinfacht auch die Wartung von Programmen: Sobald an einem Objekt eine Änderung vorgenommen wurde, spiegeln alle Programme, die es aufrufen, automatisch die Änderung wider, ohne daß neu kompiliert oder gelinkt werden muß.

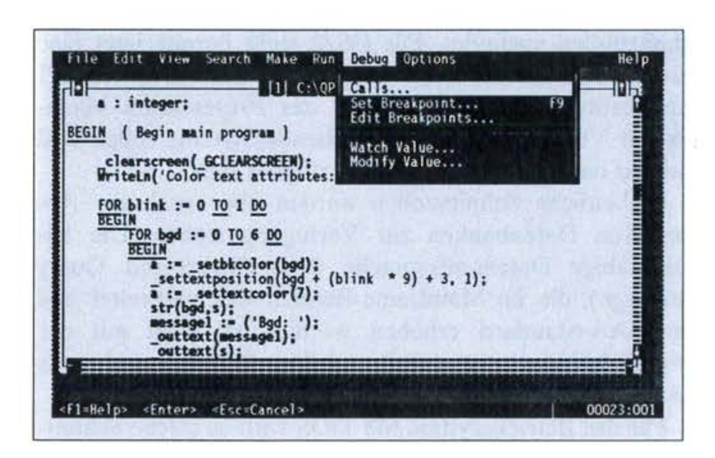


Bild 2: *QuickPascal bietet ausgezeichnete Debug-Möglichkeiten.*

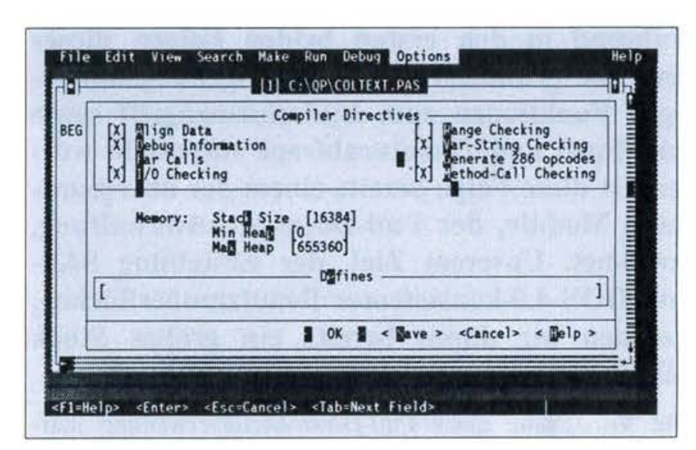


Bild 3: *In den Compilereinstellungen kann auch die Generierung von 286-Code eingestellt werden.*

Die Benutzerschnittstelle bietet eine intuitive Umgebung

QuickPascal bietet die Fähigkeit, vereinfachte Menüs, d.h. eine Untermenge der vollständigen Menüs anzuzeigen, die zum Schreiben einfacher Prorgamme ausreicht. Sobald die Programmierer mit der Sprache vertraut sind und mehr Möglichkeiten verlangen, können sie die vollständigen Menüs auswählen.

Ein intelligenter Editor und ein fortschrittlicher Debugger vereinfachen die Programmierung

Der ausgefeilte Debugger kann mehrere Quelldateien, zum Beispiel ein Hauptprogramm und seine Units, in mehreren Fenstern debuggen und er ist der einzige Pascal-Debugger für Personalcomputer, der in der Lage ist, sowohl Assembler- als auch Pascal-Code zu debuggen. Anwender können Assemblerroutinen in ihre Programme einbauen, mit der Einzelschrittfunktion die Befehle durchgehen und die Inhalte der CPU-Register anzeigen.

Online-System und gedruckte Dokumentation erfüllen die Anforderungen von Programmierern

Eine umfangreiche Dokumentation, die sowohl Online- als auch gedruckte Dokumentation enthält, bietet Anwendern eine umfassende, einfach zu verwendende Lösung für ihre Probleme mit dem Nachschlagen von Informationen.

QuickPascal wird darüber hinaus mit QuickPascal-Express geliefert, einem schnellen und freundlichen Online-Tutorial, das Benutzern die Bedienungsgrundlagen von QuickPascal (Öffnen einer Datei, Editierung, Debuggen usw.) beibringt.

Systemanforderungen und Verfügbarkeit

Microsoft QuickPascal benötigt 512 Kbyte freien Speicher, zwei doppelseitige Diskettenlaufwerke und MS-DOS (Version 2.1 oder neuer). Der Einsatz einer Microsoft Maus ist optional. QuickPascal wird ab Mitte Mai verfügbar sein.

Eine komfortable Benutzeroberfläche in C (Teil 3):

Pull-Down-Menüverwaltung

Während in den ersten beiden Folgen dieser Serie die grundlegenden und umgebungsabhängigen Funktionen zum Bildschirmzugriff sowie zur Maus- und Tastaturabfrage vorgestellt wurden, ist diese Folge bereits einem der übergeordneten Module, der Pull-Down-Menüverwaltung, gewidmet. Unserem Ziel, der Erstellung SAA- bzw. DOS-4.0-konsistenter Benutzeroberflächen, kommen wir damit bereits ein großes Stück näher.

Die Vorstellung einer Pull-Down-Menüverwaltung markiert im Rahmen dieser Serie einen wichtigen Einschnitt, verlassen wir doch den Bereich der Basismodule und wenden uns den übergeordneten, den eigentlichen SAA-Modulen zu. Damit ändert sich auch die Fragestellung, vor deren Hintergrund die einzelnen Module konzipiert und realisiert werden. Nicht mehr die Frage, wie auf Maus und Tastatur zugegriffen oder Zeichen möglichst rasch auf den Bildschirm gebracht werden können, sondern die Frage, wie das SAA-Konzept in die Gestaltung einer Benutzeroberfläche umgesetzt werden muß, stehen von nun ab im Vordergrund. Dabei gilt es zunächst zu klären, welche Bereiche das SAA-Konzept abdeckt, welche Standards es in diesen Bereichen vorgibt und welche Implikationen dies für die Erstellung von Applikationen unter MS-DOS mit sich bringt.

Das SAA-Konzept

Als IBM im Frühjahr 1987 seine neuen PS/2-Modelle vorstellte, präsentierte es der Öffentlichkeit gleichzeitig ein Konzept, daß unter dem Namen SAA (System Application Architecture) als eines der größten und ehrgeizigsten Softwareprojekte in die noch junge Geschichte der Computerindustrie eingehen wird. Das Ziel ist es, die Benutzeroberflächen, Programmierschnittstellen und Kommunikationsprotokolle der PC-, Mainframe- und Großrechner-Welt so zu vereinheitlichen, daß nicht nur ein ungehemmter Datenaustausch zwischen diesen Systemen möglich wird, sondern es für Anwender wie Programmierer keine Rolle mehr spielt, mit welchem dieser Systeme sie arbeiten.

Man darf sich SAA daher nicht als ein Computer-Programm ooder ähnlich vorstellen, sondern als einen Satz von Regeln und Protokollen, an denen sich Programmierer in der Zukunft orientieren können. Diese Regeln werden in drei Bereiche untergliedert:

Common Programming Interface (CPI)

Hier werden die Schnittstellen beschrieben, auf deren Basis ein Programm auf die Systemressourcen wie Bildschirm, Dateien, Drucker etc. zugreifen kann. Bedient sich ein Programm dieser Schnittstellen, kann es ohne großen Aufwand in eine andere SAA-Umgebung, also auf ein anderes Rechnersystem portiert werden, da es dort die gleichen Schnittstellen vorfindet. Für OS/2 steht bereits jetzt eine partielle Realisation dieser Schnittstellen in Form des API (Application Program Interface) des Presentation Managers zur Verfügung, und entsprechende Systeme sollen bald auch für die IBM-Großrechner verfügbar sein.

Einheitliche Schnittstellen werden aber auch zur Abfrage von Datenbanken zur Verfügung stehen. Die leistungsfähige Datenbanksprache SQL (Structured Query Language), die im Mainframe-Bereich weit verbreitet und zum SAA-Standard erhoben worden ist, steht mit der Extended Edition von OS/2 und dem SQL-Server bereits jetzt auch im PC-Bereich zur Verfügung.

Für das Betriebssystem MS-DOS wird es solche Schnittstellen jedoch nur in sehr eingeschränkter Form geben, da MS-DOS im Gegensatz zu OS/2 den Leistungsanforderungen des SAA-Konzepts nur bedingt gerecht wird.

Die Portierung von Programmen zwischen den verschiedenen SAA-Umgebungen wird auch dadurch unterstützt werden, daß einheitliche Hochsprachen-Compiler und Entwicklungstools in allen SAA-Umgebungen bereitgestellt und dadurch einheitliche Entwicklungsumgebungen geschaffen werden. Zu diesen Tools wird z.B. ein Application-Generator zählen, der den Programmierer interaktiv bei der Gestaltung all jener Programmteile unterstützt, die von dem CPI-Konzept betroffen werden. Portierungsprobleme, wie sie Heute beispielsweise bei der Portierung von Programmen aus der MS-DOS- in die Unix-Umgebung an der Tagesordnung sind, sollen dann der Vergangenheit angehören.

Common User Access (CUA)

Dieser Bereich des SAA-Konzepts spezifiziert den Aufbau und die Bedienung der Benutzeroberflächen, denen sich der Anwender bei der Arbeit mit einer Applikation gegenübersieht. Auch hier wurden Standards gesetzt, so daß die verschiedenen Applikationen dem Anwender in Zukunft ein einheitliches »Look and Feel« bieten werden. (Man spricht hier auch von »syntaktischer Konsistenz«). Der Schulungsaufwand bei der Einführung neuer Applikationen wird dadurch minimiert und die Akzeptanz der EDV im Kreise der Anwenderschaft insgesamt gesteigert.

Ihre Verwirklichung finden diese Regeln zum Teil im Common Programming Interface, beispielsweise im API des Presentation Managers, der dem Programmierer die Möglichkeit bietet, Fenster, Pull-Down-Menüs und Dialogboxen nach dem CUA-Standard für die Kommunikation mit dem Anwender zu nutzen. Vorbild sind hier der Apple Macintosh und Windows, die beide eine ebenso gut strukturierte wie konsistente Benutzeroberfläche bieten.

Beide jedoch operieren im Grafikmodus, während die meisten PC-Applikationen unter MS-DOS im Textmodus arbeiten. Doch auch auf den Textmodus können die Regeln des CUA-Standards übertragen werden, wie das Beispiel MS-Works zeigt, das gemeinhin als der Prototyp einer SAA-Applikation unter MS-DOS angesehen wird.

Der CUA-Standard geht jedoch über die Definition des Erscheinungsbildes der Benutzeroberfläche und ihrer Bedienung hinaus. So fordert er z.B. eine einheitliche Gestaltung und Belegung der Tastatur, damit beispielsweise der PC-Anwender die Taste [F1] auch an einem Mainframe-Terminal an der gewohnten Stelle vorfindet. Über diese »physikalische Konsistenz« hinaus fordert der CUA-Standard aber auch die »semantische Konsistenz« der verschiedenen Applikationen, die sich z.B. darin ausdrückt, daß die Betätigung der Taste [F1] immer die gleiche Programmfunktion auslöst, nämlich »Hilfe«.

Common Communications Support (CCS)

Damit ein uneingeschränkter Datenaustausch über die Systemgrenzen hinweg möglich wird, definiert der CCS-Standard die Kommunikationsprotokolle, mit denen Daten zwischen verschiedenen Rechnersystemen über Netzwerke, Telefon- und Datenleitungen etc. ausgetauscht werden. Die Funktionen im »Common Programming Interface« unterstützen diese Protokolle und erlauben es einem Programm dadurch, auf entfernte Datenbestände zuzugreifen.

Ein Beispiel dafür ist der »Database Manager« in der Extended Edition von OS/2, der die Auswertung und Verknüpfung von Datenbanken erlaubt, unabhängig, ob sie auf einem PC, innerhalb eines Netzwerkes oder auf einem Mainframe gespeichert sind.

SAA und die Pull-Down-Menüverwaltung

Für die Realisierung einer Pull-Down-Menüverwaltung nach dem SAA-Standard unter MS-DOS spielt der Bereich des »Common User Access« eine große Rolle, während das »Common Programming Interface« (mangels Masse) und der »Common Communications Support« nicht von Bedeutung sind.

Der CUA-Standard sieht eine Unterteilung des Bildschirms in drei Bereiche vor:

- die Menüleiste (action bar) in der ersten Bildschirmzeile, in der die Namen der einzelnen Menüs angezeigt werden,
- die Infozeile (function key area), die die unterste Bildschirmzeile belegt und in der Informationen, wie etwa die Belegung der Funktionstasten oder kurze Hilfstexte zu den einzelnen Menüs angezeigt werden,
- der Arbeitsbereich der Applikation (panel body), der von der Menüleiste und der Infozeile eingegrenzt wird. Seine Größe beträgt normalerweise 23 Zeilen, das hängt aber von dem jeweiligen Video-Modus ab. In Verbindung mit den erweiteren Video-Modi der EGA- und VGA-Karte nimmt er eine Größe von 41 bzw. 48 Zeilen ein. Die Applikation entscheidet selbständig über den Aufbau dieses Bildschirmbereichs und ist in dieser Hinsicht keinerlei Restriktionen durch den CUA-Standard unterworfen.

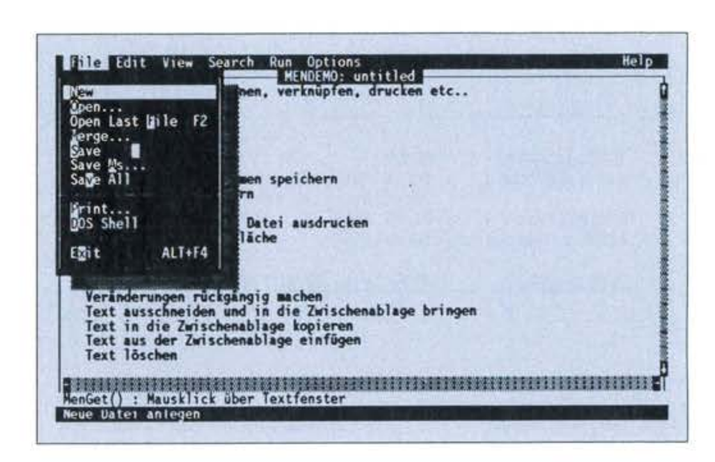


Bild 1: *So präsentiert sich das Demoprogramm MENDEMO.C*

Innerhalb dieser Folge unserer SAA-Serie spielt vor allem der Aufbau der Menüleiste eine große Rolle. Da sie im Rahmen einer SAA-Applikation die wichtigste Schnittstelle zwischen dem Anwender und der Applikation darstellt und der Auswahl *aller* Programmfunktionen dient, wird sie permanent auf dem Bildschirm angezeigt und darf nicht etwa durch einen Teil des Arbeitsbereichs überdeckt werden. Sie sollte farblich so gestaltet sein, daß sie sich von dem darauffolgenden Arbeitsbereich abhebt und die Trennung zwischen ihr und dem Arbeitsbereich dadurch bereits optisch deutlich wird.

Die einzelnen Menütitel werden nicht gleichmäßig auf die Menüleiste verteilt, sondern folgen jeweils im Abstand von 2 Leerzeichen aufeinander, wobei der erste Menütitel in der dritten Spalte beginnt. Eine Ausnahme bilden lediglich die Menütitel »Hilfe« und »Ende«, die am rechten Rand der Menüleiste angeordnet werden, um sie gegenüber den anderen Menütiteln hervorzuheben.

Während jede Applikation nach dem SAA-Standard über einen Menütitel (und damit über einen Befehl) »Hilfe« verfügen soll, muß der Befehl »Ende« nicht unbedingt als eigenständiger Menütitel in der Menüleiste aufgeführt werden. Vielmehr findet man ihn oft als letztes Untermenü im Menüpunkt »Datei«, so daß der Menütitel »Hilfe« oft das einzige Menü darstellt, daß am rechten Rand der Menüleiste zu finden ist. Die Trennung zwischen dem Hilfe-Menü und den anderen Menüs wird in manchen Applikationen zusätzlich dadurch unterstützt, daß links neben dem Hilfe-Menü ein senkrechter Strich in die Menüleiste gezogen wird. Das hier vorgestellte Menümodul stellt es ihnen frei, diesen Strich in die Menüleiste aufzunehmen oder ihn wegzulassen.

Die Reihenfolge der einzelnen Menüs innerhalb der Menüleiste gibt der SAA-Standard nicht explizit vor, empfiehlt jedoch, sie so anzuordnen, daß sich die am häufigsten benutzten Menüs am linken Rand der Menüleiste befinden. Die einzelnen Menüs sollen dabei kurze, prägnante Namen tragen und mit den Menünamen bereits verbreiteter SAA-

```
/*
[ ]-----------------------------------------------------------[ ]
 |                                                             |
 |     Include-Datei    : MEN.H                                |
 |     zur Einbindung   : der Funktionen zur Erstellung und    |
 |                        Verwaltung von Pull-Down-Menüs       |
 |     erstellt am      : 21.02.1989                           |
 |     letztes Update am: 14.03.1989                           |
 |                                                             |
 |     (Copyright)      : 1989 by MICHAEL TISCHER              |
 |                                                             |
[ ]-----------------------------------------------------------[ ]
*/

/*== interne Deklarationen für die MEN-Module ==============*/

#ifdef SAA_MEN_INTERN              /* Einbindung in MEN-Modul? */

/*== Konstanten ============================================*/

#define NO_HOTKEY ((TASTE) 0)       /* Menü hat keinen Hotkey */
#define GOT_HOTKEY( p ) ( *(p) < 32 )        /* TRUE, wenn HK */

/*-- die folgenden Konstanten beeinflußen das Erschei-    ---*/
/*-- nungsbild der Menüleiste und der Untermenüfenster    ---*/

#define MEN_STRICH   FALSE        /* Strich neben Menü rechts */
#define MEN_SCHATTEN TRUE        /* Schatten um das Untermenü */
#define MEN_HORAT    EINRA          /* horizontaler Menürahmen */
#define MEN_VERAT    EINRA            /* vertikaler Menürahmen */
#define MEN_ZEILE    0           /* Bildschirmzeile Hauptmenü */
#define MEN_LINKS    2              /* Startspalte erstes Menü */
#define MEN_RECHTS   2            /* Abstand vom rechten Rand */
#define MEN_ABSTAND  2           /* Spaltenabstand Hauptmenüs */
#define MEN_TOG_CHAR "√"

#define MEN_SM_LINKS  1        /* Rahmen -> Menü im Untermenü */
#define MEN_SM_RECHTS 1        /* Menü -> Rahmen im Untermenü */
#define MEN_SM_START  2      /* Untermenürahmen -> Hauptmenü */

#define MEN_HORA EINRA  /* horiz. Rahmen um SM: EINRA/DOPRA */
#define MEN_VERA EINRA   /* veti. Rahmen um SM: EINRA/DOPRA */
#define MEN_HM_KEY F10      /* Taste zur Aktivierung des Hm. */

#define MEN_R_START  ( VioGetCols() - 1 - MEN_RECHTS )
#define MEN_BREITE   ( MEN_SM_LINKS + MEN_SM_RECHTS + 2)
#define MEN_G_BREITE ( MEN_BREITE + 2 )

/*-- Mauscursor für an- und ausgeschaltete Menüs -----------*/
#define MEN_STD_CM MouPtrMask( PTRSAMECHAR, PTRINVCOL )
#define MEN_DIS_CM MouPtrMask( PTRDIFCHAR('X') , PTRINVCOL )
#define MEN_STD_CC \
 MouPtrMask( PTRSAMECHAR, PTRDIFCOL( COL( SCHWARZ, WEISS ) ) )
#define MEN_DIS_CC \
 MouPtrMask(PTRDIFCHAR('X') , PTRDIFCOL( COL(SCHWARZ,WEISS)) )

#define MEN_STD_CUR ( VioIsColor() ? MEN_STD_CC : MEN_STD_CM )
#define MEN_DIS_CUR ( VioIsColor() ? MEN_DIS_CC : MEN_DIS_CM )

/*-- Konstanten sparen Platz im Listing --------------------*/
#define RG register      /*für Deklaration von Register-Var.*/
#define SB ( sizeof( BEREICH ) )
#define RC ( farben.rcol )                   /* Rahmenfarbe */

/*-- interne MEV-Codes, dringen nicht nach außen -----------*/
#define MEV_ESCAPE    128               /* Ende durch ESCAPE */
#define MEV_HAUPTMEN 129           /* Hauptmenü ausgewählt */
#define MEV_MOVE      130           /* neues Menü angewählt */
#define MEV_ENDE      131     /* zurück auf unterste Ebene */
#define MEV_NO        132        /* kein Ereignis verfügbar */
#define MEV_POINT     133        /* in Point-Modus wechseln */
#define MEV_DRAG      134         /* in Drag-Modus wechseln */

/*-- Return-Codes für MenIGetBereich() ---------------------*/
#define BER_MAUS 0                           /* Mausbereich */
#define BER_HMEN 1                             /* Hauptmenü */
#define BER_SMEN 2                             /* Untermenü */
#define BER_NO   3           /* kein registrierter Bereich */
```

Listing 1: *MEN.H*

```
/*== Makros ================================================*/

#define MenIsOk(m)          ( ((hmenptr+m)->disable & 1) == 0 )
#define SubMenIsOk(p,m) \
   ( *(p->mena+m)!=MEN_TRENNER && (p->disable & (1 << m))==0 )

/*== Strukturen ============================================*/

typedef struct                      /* Rahmenzeichen für Menüs */
{
 char *ol,                                /* obere linke Ecke */
      *or,                               /* obere rechte Ecke */
      *ul,                               /* untere linke Ecke */
      *ur,                              /* untere rechte Ecke */
      hori,                            /* horizontaler Strich */
      verti;                             /* vertikalter Strich */
} RAHMEN;

#endif                 /* Ende der konditionalen Kompilierung */

/*== dieser Teil ist auch allen Modulen zugänglich, die ====*/
/*== mit den Funktionen aus den MEN-Modulen arbeiten     ====*/

/*== Typedefs ==============================================*/

#ifndef BYTE                        /* BYTE bereits definiert? */
  typedef unsigned char BYTE;            /* Nein, definieren */
#endif

typedef void (* MENFKT)( BYTE );   /* Menü-Wechsel-Funktion */

typedef unsigned long ULONG;                    /* 32-Bit-Datum */

typedef struct                   /* Beschreibt einen Menüpunkt */
{
 BYTE    anzahl,                       /* Anzahl der Untermenüs */
         knr;                                    /* Kennnummer */
 char    **mena;    /* Ptr auf Vektor mit Ptr auf Menünamen */
 TASTE   *awb;    /* Pointer auf Vektor mit Auswahlbuchst. */
 BEREICH *smber;      /* Bereiche der einzelnen Untermenüs */
 BYTE    rx1, ry1,              /* Eckkordinaten des Rahmens */
         rx2, ry2;
 ULONG   toggles,   /* zeigt, welche Menüs getoggelt werden */
         tontoff,    /* zeigt, ob diese Menüs on / off sind */
         disable;      /* zeigt, welche Menüs disabled sind */
} HMDES;

typedef char * MENUE[];      /* Vektor mit Ptr auf Menünamen */


typedef struct                                 /* Menü-Farben */
{
 BYTE sn,               /* Menü schlummert, normales Zeichen */
      sh,                           /* Menü schlummert, Hotkey */
      an,               /* Menü ist aktiv, normales Zeichens */
      ah,                            /* Menü ist aktiv, Hotkey */
      disable;            /* Menü darf nicht angewählt werden */
} MC;                                           /* Menu Color */


typedef struct /* Farben für Hauptmenü, Untermenüs & Rahmen */
{
 MC   hm,                          /* Farben für das Hauptmenü */
      sm;                          /* Farben für die Untermnüs */
 BYTE rcol;                     /* Rahmenfarbe für Untermenüs */
} HURC;


typedef struct                                  /* Menü-Event */
{
 BYTE code;                           /* spezifiziert das Event */
 union                /* weitere Informationen über das Event */
  {
   TASTE key;                           /* für Event MEV_TASTE */
   BYTE  bereich;                        /* für Event MEV_MAUS */
   BYTE  befehl;                        /* für Event EV_BEFEHL */
  } d;                                            /* das Datum */
} MEV;                                           /* Menü EVent */
```

Listing 1: *(Fortsetzung)*

```
/*== Konstanten ==============================================*/

#ifndef TRUE            /* TRUE und FALSE bereits definiert? */
  #define TRUE  1                                    /* Nein */
  #define FALSE 0
#endif

#define MARK_CHAR   '^'     /* geht dem Auswahlbuchst. voran */
#define KEIN_MENU   255             /* kein Menü angewählt */
#define MEN_TRENNER ( (char *) 0 )   /* Trennstrich im Menü */

/*-- Event-Codes, wie sie MenGet() zurückliefert ------------*/

#define MEV_TASTE  1           /* es wurde eine Taste betätigt */
#define MEV_MAUS   2  /* linker Mausknopf über Bereich bet. */
#define MEV_BEFEHL 3                  /* ein Menü ausgewählt */

/*== Makros ==================================================*/

/*-- MD hilft beim Anlegen des Menü-Vektors -----------------*/
#define MD( mv, hnr ) { ELVEK(mv)-1, hnr, mv, (TASTE *) 0,\
                        (BEREICH *) 0, 0, 0, 0, 0, 0, 0, 0 }

/*== Funktionsdeklarationen über Makros ====================*/

#define MenSetSchattenOn()      ( schatten = TRUE )
#define MenSetSchattenOff()     ( schatten = FALSE )
#define MenSetRahmen( h, v )    horat=h, verat=v
#define MenSetHMKey( t )        ( hmkey = t )
#define MenGetHMKey()           ( hmkey )
#define MenSetFkt( f )          ( menfkt = f )
#define MenGetFkt()             ( menfkt )
#define MenSetToggleOn( nr )    MenSetFlag( nr, TRUE, TRUE )
#define MenSetToggleOff( nr )   MenSetFlag( nr, TRUE, FALSE )
#define MenEnableMen( nr )      MenSetFlag( nr, FALSE, FALSE )
#define MenDisableMen( nr )     MenSetFlag( nr, FALSE, TRUE )
#define MenGetAktMen()          ( aktmen )
#define MenWinClose( x )        VioWinClose( x )
#define MenSetCol( x )          ( farben = x )

/*== externe Variablen =======================================*/

extern TASTE  hmkey;
extern MENFKT menfkt;
extern BYTE   aktmen,
              horat,
              verat,
              schatten;
extern HURC   farben;

/*== Funktions-Deklarationen =================================*/

void MenRegister      ( HMDES * hmptr, BYTE anz, BYTE rechts );
void MenSetFelder     ( BYTE anz, BEREICH *bptr );
void MenPrintMen      ( void );
MEV  MenGet           ( void );
void MenWinOpen       ( BYTE x1, BYTE y1, BYTE x2, BYTE y2,
                        BYTE hora, BYTE vera, BYTE rcol );
void MenDefToggle     ( BYTE menr, BYTE init );
void MenSetFlag       ( BYTE menr, BYTE flag, BYTE wert );
void MenReplaceSubMen( BYTE nr, HMDES *mptr );
void MenReplaceName   ( BYTE nr, char *name );
void MenConvNameBack  ( char * nptr );
```

Listing 1: *(Ende)*

Applikationen möglichst übereinstimmen. Jedes Menü muß dabei über einen sogenannten Hotkey verfügen, der vom Menü-Modul farblich hervorgehoben wird und über den es in Verbindung mit der [Alt]-Taste aufgerufen werden kann.

Normalerweise sollte es sich bei diesem Hotkey um den jeweils ersten Buchstaben eines Menüs handeln, doch setzt dies voraus, daß die einzelnen Menüs über jeweils unterschiedliche Anfangsbuchstaben verfügen. Ist dies nicht möglich, kann auch ein anderer Buchstabe als Hotkey gewählt werden, doch sollte dies eine Ausnahme bleiben.

Jedes Menü kann - muß aber nicht - über ein Untermenü verfügen, das bei Auswahl des Menüs angezeigt wird, wobei es einen Teil des Arbeitsbereichs überdeckt. Um die Abgrenzung zwischen dem Untermenü und dem Arbeitsbereich hervorzuheben, wird das Menü von einem Rahmen umgeben, der nach dem SAA-Standard aus doppelten Linien bestehen soll. In der Praxis ziehen viele Applikationen jedoch einen einfachen Rahmen vor. Das hier vorgestellte Menü-Modul läßt Ihnen in dieser Hinsicht freie Hand; Sie können die Art des Rahmen selbst auswählen.

Häufig umgeben SAA-Applikationen das Untermenü am unteren und am rechten Rand mit einem Schatten, der einen 3-D-Effekt erzeugt und den Eindruck erweckt, das Untermenü befände sich über dem Arbeitsbereich. Auch das hier vorgestellte Menü-Modul bietet Ihnen auf Wunsch die Möglichkeit, die verschiedenen Untermenüs mit einem Schatten zu versehen, wobei die Art des Schattens vom jeweils aktuellen Video-Modus abhängt. Im Color-Modus wird der Schatten durch Manipulation der Attribute der Zeichen erzeugt, die sich innerhalb des Schattens befinden. Es wird dazu jeweils das Intensitätsbit der Vorder- und Hintergrundfarbe gelöscht, bzw. die Farbe auf 0 (schwarz) gesetzt, wenn das Intensitätsbit bereits gelöscht war. Dadurch entsteht ein sehr schöner Effekt, wie man ihn beispielsweise bei MS-Works betrachten kann.

Leider bietet der Monochrom-Modus mit seiner sehr eingeschränkten Anzahl an Farben bzw. Attributen diese Möglichkeiten nicht, so daß man sich hier eines anderen Verfahrens bedienen muß: Anstelle einer Attribut-Veränderung werden in diesem Modus die Zeichen, die durch den Rahmen überdeckt werden sollen, durch das Zeichen mit dem ASCII-Code 177 (▒) ersetzt.

Das Untermenü-Fenster wird so angeordnet, daß das Hauptmenü und die einzelnen Untermenüs in der gleichen Bildschirmspalte beginnen. Die Breite des Fensters orientiert sich dabei an der Breite des längsten Untermenüs, wobei zwischen den Menüs und dem Rahmen ein Freiraum von jeweils einem Leerzeichen bleibt. Dieser Freiraum wird bei Untermenüs, die als Schalter fungieren, genutzt, um den Status des Schalters anzuzeigen. Steht der Schalter auf »An«, wird anstelle eines Leerzeichens, das »√«-Zeichen zwischen dem linken Rand des Untermenüs und dem Rahmen ausgegeben.

Auch für die Anordnung der Untermenüs gilt, daß die am häufigsten verwendeten Menüs leicht zu erreichen und daher im oberen Teil des Untermenü-Fensters zu finden sein sollen. Besteht das Untermenü aus mehreren Gruppen logisch zusammenhängender Befehle, können diese Gruppen durch einen horizontalen Strich voneinander getrennt werden.

Eine weitere Parallele zu den Hauptmenüs ist die Verwendung von Hotkeys, die bei den Untermenüs allerdings nicht so zwingend vorgeschrieben ist, wie bei Hauptmenüs.

```
/**************************************************************/
/*                  M E N D E M O . C                          */
/*-------------------------------------------------------------*/
/*    Aufgabe        : Demonstriert die Arbeit mit den Funk-*/
/*                     tionen aus dem Menü-Modul.             */
/*-------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                         */
/*    entwickelt am  :  1.03.1989                              */
/*    letztes Update : 14.03.1989                              */
/*-------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] MENDEMO.C MEN.C        */
/*                     MENUTIL.C KBM.C VIO.C                   */
/**************************************************************/

/*-- Include-Dateien einbinden ------------------------------*/

#include "vio.h"         /* Zugriff auf die Video-Funktionen */
#include "kbm.h"           /* eigene Include-Datei einbinden */
#include "men.h"

/*-- Konstanten zur Arbeit mit den Menüs -------------------*/

#define MEN_FILE        0 /* Basisnummer File-Menü          */
#define MEN_EDIT       13 /* Basisnummer Edit-Menü          */
#define MEN_VIEW       21 /* Basisnummer View-Menü          */
#define MEN_SEARCH     28 /* Basisnummer Search-Menü        */
#define MEN_RUN        37 /* Basisnummer Run-Menü           */
#define MEN_OPTIONSNF  44 /* Basisnr. Options-Not-Full-Menü */
#define MEN_OPTIONSF   47 /* Basisnummer Options-Full-Menü  */
#define MEN_HELP       53 /* Basisnummer Help-Menü          */

/*-- Konstanten für die Unterscheidung der Farben---------- */

#define WB ( VioIsColor() ? COL( WEISS, BLAU ) : HNORMAL )
#define WV ( VioIsColor() ? COL( WEISS, VIOLETT ) : INVERS )
#define WS ( VioIsColor() ? COL( WEISS, SCHWARZ ) : INVERS )
#define SV ( VioIsColor() ? COL(SCHWARZ, VIOLETT) : UNDERLINE)
#define HRB ( VioIsColor() ? COL( HROT, BLAU ) : INVERS )
#define MCOL PTRDIFCOL( COL( ROT, WEISS ) )

/*-- Konstanten, sparen Platz im Listing -------------------*/

#define MSG( s, t ) VioPrint( VL(s), VU(-1), SV, FALSE, t )
#define MSGF( s, t, a ) \
                   VioPrintf( VL(s), VU(-1), SV, FALSE, t, a )
#define MPM( p1, p2 ) MouPtrMask( p1, p2 )

/*-- globale Variablen --------------------------------------*/

char *cstr[] =                      /* Hilfstexte für die Menüs */
 {
  "Dateien anlegen, öffnen, verknüpfen, drucken etc..",
  "Neue Datei anlegen",
  "Neue Datei öffnen",
  "Letzte Datei öffnen",
  "Dateien verknüpfen",
  "Datei speichern",
  "Datei unter neuem Namen speichern",
  "Alle Dateien speichern",
  "",
  "Datei oder Teile der Datei ausdrucken",
  "Aufruf der DOS-Oberfläche",
  "",
  "Programm beenden",
  "Datei editieren",
  "Veränderungen rückgängig machen",
  "Text ausschneiden und in die Zwischenablage bringen",
  "Text in die Zwischenablage kopieren",
  "Text aus der Zwischenablage einfügen",
  "Text löschen",
  "",
  "Datei darf nicht verändert werden",
  "Auswahl des Bildschirminhalts",
  "Quellprogramm betrachten",
  "Include-Datei betrachen",
  "",
  "Ausgabebildschirm betrachten",
  "Fenster vergrößern",
```

Listing 2: *MENDEMO.C*

Zwar sollten die meisten Untermenüs über einen Hotkey verfügen und dieser nach Möglichkeit der erste Buchstabe in dem jeweiligen Menü sein, doch darf ein Untermenü auch über Menüs verfügen, die nicht über einen Hotkey erreicht werden können.

Dies gilt insbesondere für Menüs, die unter Umgehung des Hauptmenüs über eine Funktionstaste aufgerufen werden können. Diese Tastenkombination sollte im Menünamen vermerkt sein, damit der Anwender bei der Betrachtung des Untermenüs von dieser Möglichkeit Notiz nehmen kann. Der SAA-Standard sieht dabei vor, daß die Tastenkombination zum Aufruf der verschiedenen Menüs rechtsbündig mit dem rechten Rand des breitesten Untermenüs abschließt, wobei zwischen dem Menünamen und der Tastenkombination jedoch ein Freiraum von mindestens zwei Leerzeichen verbleiben soll.

In den Menünamen sollte auch die Information einfließen, ob der Aufruf des Menüs unmittelbar zur Ausführung des damit verbundenen Befehls führt, oder ob zunächst eine Dialogbox (ein Fenster mit Eingabemasken) geöffnet wird, in dem der Anwender einzelne Parameter einstellen kann, die die Ausführung des Befehls beeinflussen. Im letzteren Fall läßt man dem Menünamen drei Punkte folgen, die darauf hinweisen, daß weitere Angaben zur Ausführung des Befehls gemacht werden müssen.

Auswahl von Menüs

Neben dem Erscheinungsbild der Menüleiste und der Untermenü-Fenster legt der SAA-Standard auch die Art und Weise fest, wie Menüs und Untermenüs ausgewählt werden können. Maus und Tastatur werden dabei gleichermaßen als Eingabegeräte berücksichtigt, wobei der Maus in grafikorientierten SAA-Umgebungen fast noch eine wichtigere Rolle zukommt als der Tastatur.

Der Einstieg in das Hauptmenü über die Tastatur erfolgt in SAA-Umgebungen grundsätzlich über die Funktionstaste [F10]. Daraufhin wird das erste Menü in der Menüleiste als das aktuelle Menü farblich hervorgehoben. Mit Hilfe der Cursortasten [→] und [←] sowie [Backspace] und der [Leertaste] kann man dann die einzelnen Menüs auswählen, wobei die Cursorbewegung nicht beim letzten bzw. ersten Menü in der Menüleiste halt macht, sondern automatisch wieder zum ersten bzw. letzten Menü springt (wrap-around). Ein direkter Sprung zum ersten bzw. letzten Menü der Menüleiste ist über die Tasten [Home] bzw. [End] möglich.

Die Anwahl des aktuellen Menüs erfolgt durch die Betätigung der [Return]-Taste oder durch die Betätigung eines Hotkeys, wobei dann ungeachtet des aktuellen Menüs das Menü ausgewählt wird, dessen Hotkey eingegeben wurde. Unter Umgehung der [F10]-Taste kann ein Menü auch direkt angewählt werden, indem sein Hotkey in Verbindung mit der [Alt]-Taste betätigt wird.

Verfügt ein Menü über ein Untermenü, wird nach seiner Auswahl ein Untermenü-Fenster geöffnet, in dem die einzelnen Untermenüs dargestellt werden. Automatisch wird das erste Untermenü hervorgehoben und als das aktuelle Menü gekennzeichnet. Die Auswahl des gewünschten Menüs ist mit Hilfe der Tasten [↑], [↓], [Home] und [End] und anschließender Bestätigung über die [Return]-Taste möglich. Aber auch hier kann man sich die Auswahl erleichtern, indem man den Hotkey des gewünschten Menüs betätigt.

Der Wechsel in ein anderes Hauptmenü kann innerhalb eines Untermenüs mit Hilfe der Tasten [→] und [←] vollzogen werden. Das aktuelle Untermenü-Fenster wird in diesem Fall geschlossen und ein Untermenü-Fenster für das neue Hauptmenü geöffnet, sofern dieses über Untermenüs verfügt. Wie auch das Hauptmenü, kann ein Untermenü-Fenster durch Betätigen der [Esc]-Taste verlassen werden, ohne daß ein Menü ausgewählt wurde.

Für die Auswahl eines Menüs mit Hilfe der Maus stehen Ihnen zwei Auswahlmodi zur Verfügung, die als »press and hold« bzw. »press and release« bezeichnet werden. Beiden gemeinsam ist die Art und Weise wie eines der Hauptmenüs ausgewählt wird: Der Mauscursor wird auf den Namen des jeweiligen Menüs in der Menüleiste bewegt und dann der linke Mausknopf niedergedrückt. Sofern das angewählte Menü über Untermenüs verfügt, erscheint daraufhin automatisch das zugehörige Untermenü-Fenster. In welchem der beiden Modi die weitere Auswahl erfolgt, hängt nun davon ab, wann und wo Sie den linken Mausknopf wieder loslassen. Geschieht dies unmittelbar nach dessen Betätigung, also noch über den Menünamen, dann wird der »press and release«-Modus aktiviert. Im Untermenü-Fenster wird das erste Untermenü farblich hervorgehoben, um es als das »aktuelle« Untermenü zu markieren. Ein Untermenü können Sie in diesem Modus auswählen, indem Sie den Mauscursor über das gewünschte Menü bewegen und es »anklicken«, also den linken Mausknopf kurz niederdrücken und ihn dann wieder loslassen.

Ein anderes Hauptmenü können Sie aktivieren, indem Sie den Mauscursor wiederum auf dessen Name in der Menüleiste bewegen und ihn dann anklicken. Verlassen können Sie die Menüauswahl, indem Sie den linken Mausknopf über einem beliebigen anderen Bildschirmbereich niederdrücken und wieder loslassen.

Der rechte Mausknopf spielt in diesem, wie auch im »press and hold«-Modus für die Menüauswahl keine Rolle, doch dafür können Sie im »press and release«-Modus die Menüauswahl auch jederzeit mit den Cursortasten fortsetzen, wie es oben beschrieben wurde.

In den »press and release«-Modus gelangen Sie, indem Sie den linken Mausknopf nach der Auswahl eines Hauptmenüs nicht wieder loslassen, sondern ihn weiterhin niederdrücken. Bewegen Sie den Mauscursor dann über eines der Untermenüs wird dieses Untermenü automatisch farblich hervorgehoben und so als das aktuelle Menü markiert, das Sie durch Loslassen des Mausknopfs auswählen können.

```
  "Abzubildendes Fenster auswählen",
  "Text suchen und verändern",
  "Text suchen",
  "Markierten Text suchen",
  "Letzten Suchvorgang wiederholen",
  "Suchen und Ersetzen",
  ""
  "Funktionen suchen",
  "",
  "Nächster Fehler",
  "Vorhergehender Fehler",
  "Programm starten, Programmablauf verfolgen",
  "Neustart",
  "Programmausführung fortsetzen",
  "Programmausführung bis zur Cursorposition fortsetzen",
  "Funktionsausführung verfolgen",
  "Funktionsaufruf überspringen",
  "Programmablauf im Zeitlupentempo",
  "Einstellung verschiedenster Parameter",
  "Bildschirmparameter einstellen",
  "Alle Options-Befehle anzeigen",
  "Einstellung verschiedenster Parameter",
  "Bildschirmparameter einstellen",
  "MAKE-Parameter einstellen",
  "Parameter für die Programmverfolgung einstellen",
  "Verzeichnisse für Icnlude-Dateien und Libraries setzen",
  "Nur eingeschränkte Options-Befehle anzeigen",
  "Hilfsinformationen in allen Lebenslagen",
  "Hilfsindex anzeigen",
  "Übersicht über Hilfsthemen",
  "Hilfeinformationen im Rahmen des aktuellen Kontexts",
  "Über die Arbeit mit den Hilfefunktionen"
 };

/*-- Beschreibung der einzelnen Haupt- und Untermenüs ------*/

MENUE file = {          /* das File-Menü und seine Untermenüs */
            "^File",
            "^New",
            "^Open...",
            "Open Last ^File  F2",
            "^Merge...",
            "^Save",
            "Save ^As...",
            "Sa^ve All",
            MEN_TRENNER,
            "^Print...",
            "^DOS Shell",
            MEN_TRENNER,
            "E^xit          ALT+F4"
           };

MENUE edit = {                                     /* Edit-Menü */
            "^Edit",
            "Undo  Alt+Backspace",
            "Cut        Shift+Del",
            "Copy        Ctrl+Ins",
            "^Paste      Shift+Ins",
            "Clear            Del",
            MEN_TRENNER,
            "^Read Only"
           };

MENUE view = {                                     /* View-Menü */
            "^View",
            "^Source",
            "^Include",
            MEN_TRENNER,
            "^Output Screen   F4",
            "^Maximize  Ctrl+F10",
            "^Windows..."
           };

MENUE search = {                                 /* Search-Menü */
              "^Search",
              "^Find...",
              "^Selected Text    Shift+\\",
              "^Repeat Last Find        F3",
```

Listing 2: *(Fortsetzung)*

```
                    "^Change",
                    MEN_TRENNER,
                    "Function...",
                    MEN_TRENNER,
                    "Next Error       Shift+F3",
                    "Previous Error   Shift+F4"
                   };

MENUE run = {                                       /* Run-Menü */
              "^Run",
              "^Restart          Shift+F5",
              "^Go                     F5",
              "^Continue to Cursor     F7",
              "^Trace Into             F8",
              "^Step Over             F10",
              "^Animate",
             };

MENUE optionsnf = {                   /* Options-Not-Full-Menü */
                    "^Options",
                    "^Display...",
                    "^Full Menus"
                   };

MENUE optionsf = {                        /* Options-Full-Menü */
                   "^Options",
                   "^Display...",
                   "^Make...",
                   "^Run / Debug...",
                   "^Environment",
                   "^Full Menus"
                  };

MENUE help = {                                     /* Help-Menü */
               "^Help",
               "^Index",
               "^Contents",
               "^Topic:               F1",
               "^Help On Help   Shift+F1",
              };

HMDES of  = MD( optionsf, MEN_OPTIONSF );
HMDES onf = MD( optionsnf, MEN_OPTIONSNF );

HMDES hauptmenue[] = {              /* beschreibt das Hauptmenü */
                      MD( file,      MEN_FILE ),
                      MD( edit,      MEN_EDIT ),
                      MD( view,      MEN_VIEW ),
                      MD( search,    MEN_SEARCH ),
                      MD( run,       MEN_RUN ),
                      MD( optionsnf, MEN_OPTIONSNF ),
                      MD( help,      MEN_HELP )
                     };


/*************************************************************
*  NA : ausgewählter Befehl nicht implementiert              *
*************************************************************/

void NA( void )
{
 static char *hz[] =                   /* Textzeilen im Fenster */
  {
   "ACHTUNG!",
   "",
   "Der ausgewählte Befehl wurde noch nicht",
   "implementiert!"
   "",
   "                           ┌────────┐",
   "                           │   OK   │",
   "                           │(RETURN)│",
   "                           └────────┘"
  };


 BEREICH bereich;        /* Bereich, über dem geklickt wurde */
 int event;                                 /* Ereignismaske */
 BYTE i;                                 /* Schleifenzähler */
```

Listing 2: *(Fortsetzung)*

Auch in diesem Modus ist der Wechsel in ein anderes Hauptmenü möglich, indem Sie den Mauscursor einfach auf den Namen dieses Menüs in der Menüleiste bewegen und dabei weiterhin den linken Mausknopf niederdrücken. Lassen Sie ihn dann über dem Menünamen wieder los, wechseln Sie automatisch in den »press and release«-Modus. Aber auch die Umschaltung von dem »press and release«- in den »hold and release«-Modus ist jederzeit möglich, indem Sie den linken Mausknopf über einem Untermenü niederdrücken, ihn dann aber nicht wieder loslassen, sondern den Mauscursor zu dem gewünschten Menü bewegen.

Möchten Sie die Menüauswahl im »press and hold«-Modus verlassen, ohne ein Untermenü auszuwählen, genügt es, den Mauscursor über eine beliebige Bildschirmposition außerhalb der Menüleiste und des aktuellen Untermenü-Fensters zu bewegen und den linken Mausknopf dann loszulassen. Eine Menüauswahl mit Hilfe der Cursortasten ist im »press and hold«-Modus nicht möglich.

Eine Menüverwaltung nach dem SAA-Standard

Ihre Umsetzung in die MS-DOS-Umgebung finden die Regeln des CUA-Standards in der Menüverwaltung, deren Listing auf diesen Seiten abgedruckt ist. Ihren Kern bilden die beiden Module `MEN.C` (*Listing 3*) und `MENUTIL.C` (*Listing 4*), die Ihren Programmen die Funktionen zum Einrichten von Menüs und deren Auswahl in der oben beschriebenen Art und Weise zur Verfügung stellen. Daß es sich im Gegensatz zu den bisherigen Folgen dieser Serie um zwei Module handelt, hängt einfach mit der inneren Komplexität und Größe dieser Module zusammen, die es nicht sinnvoll erscheinen läßt, alle Funktionen in einem Modul zu implementieren. Die Funktionen der Menüverwaltung, die von Ihren Programmen aus aufgerufen werden können, befinden sich dabei in dem Modul `MEN.C` während das Modul `MENUTIL.C` Hilfsfunktionen enthält, die von den Funktionen im `MEN`-Modul aufgerufen werden.

Zur Deklaration der Funktionen aus dem `MEN`-Modul steht wie gewohnt eine Include-Datei `MEN.H` (*Listing 1*) zur Verfügung, die auch von den beiden Modulen selbst verwendet wird, um die internen Datenstrukturen und Konstanten zu deklarieren. Ein Demoprogramm mit dem sinnreichen Namen `MENDEMO.C` (*Listing 2*) demonstriert die Arbeit mit den Funktionen aus dem `MEN`-Modul und stellt ein praktisches Gerüst für die Entwicklung der Benutzeroberfläche eigener SAA-Applikationen dar.

Im wesentlichen - auch dies ein Unterschied gegenüber den bisher vorgestellten Modulen - sind es nur vier Funktionen, die Sie zur Einrichtung der Menüstruktur und Abfrage des ausgewählten Menüs benötigen. Die erste dieser Funktionen trägt den Namen `MenRegister()` und bereitet das `MEN`-Modul auf die Arbeit mit der von Ihnen definierten Menüstruktur vor. Diese Struktur muß innerhalb Ihres Programms mit Hilfe verschiedener Datentypen angelegt werden, die in der Include-Datei `MEN.H` definiert werden.

Der Datentyp MENUE dient dabei zur Anlage eines Hauptmenüs und seiner Untermenüs, wobei MENUE im Prinzip nichts anderes darstellt, als einen Vektor mit Pointern auf C-Strings, die den Namen des Hauptmenüs und seiner Untermenüs repräsentieren. Betrachten Sie dazu das folgende Beispiel:

```
MENUE datei = {
                "^Datei",
                "^Öffnen...",
                "^Speichern",
                MEN_TRENNER,
                "^Ende"
              };
```

Hier werden das DATEI-Menü und seine vier Untermenüs definiert. Grundsätzlich stellt der erste Eintrag einer solchen Definition den Namen des Hauptmenüs dar, der später in der Menüleiste erscheint. Die darauffolgenden Einträge (sofern vorhanden) repräsentieren dann die Untermenüs dieses Hauptmenüs. Der Hotkey eines Menüs wird innerhalb dieses Vektors dadurch angegeben, daß dem jeweiligen Buchstaben innerhalb des Menünamens ein »^«-Zeichen vorangestellt wird, das bei der Ausgabe des Namens nicht auf dem Bildschirm erscheint. Fehlt dieses Zeichen, ist das entsprechende Menü nicht über einen Hotkey zu erreichen.

Das obige Menü enthält einen besonderen Eintrag, die Konstante MEN_TRENNER. Sie wird innerhalb der Include-Datei MEN.H definiert und repräsentiert eine horizontale Linie innerhalb des Untermenü-Fensters, die logisch zusammenhängende Gruppen von Untermenüs trennt. Es können durchaus mehrere dieser Trennlinien in einem Untermenü definiert werden, doch darf die Gesamtzahl der Untermenüs (inklusive der Trennlinien) 15 nicht überschreiten.

Die verschiedenen Menüs, die in der oben beschriebenen Art und Weise definiert werden, müssen für die Verarbeitung durch das MEN-Modul zu einem Vektor, bestehend aus Variablen vom Typ HMDES, zusammengefaßt werden. Stellen DATEI, BERECHNEN, OPTIONEN und HILFE einzelne Menüs dar, müssen sie folgendermaßen zusammengefaßt werden:

```
HMDES hauptmenue[] = {
                       MD( datei, Ø ),
                       MD( berechnen, 25),
                       MD( optionen, 5Ø),
                       MD( hilfe, 75)
                     };
```

Die Reihenfolge, in der die einzelnen Menüs innerhalb dieses Vektors aufgeführt werden, spiegelt die Reihenfolge wieder, in der sie später innerhalb der Menüleiste erscheinen. Initialisiert werden die einzelnen Einträge innerhalb dieses Vektors mit Hilfe des Makros MD (definiert in MEN.H), das eine Variable vom Typ HMDES anlegt. Zwar könnte eine solche Struktur auch von Hand angelegt und als Element dieses Vektors angegeben werden, doch enthält eine solche Struktur viele Informationen, die erst beim Aufruf der Funktion MenRegister( ) initialisiert werden.

```
 MouPushPara();                         /* Mausparameter sichern */
 MenWinOpen( 2Ø, 7, 63, 17, DOPRA, EINRA, WV );
 MouHideMouse();                        /* Mauscursor verstecken */
 VioClear( VL(1), VO(1), VR(-1), VU(-1), WV);

 /*-- Text im geöffneten Fenster ausgeben -----------------*/
 for (i=Ø; i < ELVEK(hz); ++i)
   VioPrint( VL(2), VO(2+i), WV, FALSE, hz[i] );

 bereich.x1 = VR(-14);        /* Position des OK-Felds setzen */
 bereich.y1 = VU( -4);
 bereich.x2 = VR( -3);
 bereich.y2 = VU( -1);
 bereich.ptr_mask= VioIsColor() ? MPM(PTRDIFCHAR(251), MCOL)
                              : MPM(PTRDIFCHAR(251), PTRINVCOL );
 MouSetBereich( 1, &bereich );     /* das OK-Feld definieren */
 MouShowMouse();                /* Maus-Cursor wieder einblenden */
 i = TRUE;                  /* die Schleife noch nicht beenden */

 do                                       /* Eingabeschleife */
  {                 /* auf linken Mausknopf oder Taste warten */
   event = KbmEventWait( EV_LEFT_PRESS | EV_KEY_AVAIL );
   if ( event & EV_LEFT_PRESS )           /* linker Mausknopf? */
    {                                                 /* Ja */
     i = ( MouGetBereich() != Ø  );          /* Bereich o.k.? */
     KbmEventWait( EV_LEFT_REL );   /* auf Loslassen warten */
    }
   else             /* linker Mausknopf nicht niedergedrückt */
    i = ( KbdGetKey() != CR );          /* Ja, Taste = RETURN? */
  }
 while ( i );

 MouHideMouse();                        /* Maus-Cursor wegblenden */
 MenWinClose( TRUE );                 /* Fenster wieder schließen */
 MouPopPara();                       /* Mausparameter zurückholen */
 MouShowMouse();                   /* Mauscursor wieder anzeigen */
}

/***********************************************************
*  TFKT : wird beim Menüwechel von MenGet() aufgerufen     *
***********************************************************/

void tfkt( BYTE nr )
{
 static char menstr[]="<F1=Help> <F1Ø=Menu> <ALT+..=Command>";

 /*-- Info-Zeile löschen und dann Hilfstext ausgeben -------*/

 VioClear( Ø, VioGetLines()-1, VioGetCols()-1,
          VioGetLines()-1, WV );
 VioPrint( 1, VioGetLines()-1, WV, FALSE,
          ( nr == KEIN_MENU) ? menstr : cstr[nr] );
}

/***********************************************************
* H A U P T P R O G R A M M                                *
***********************************************************/

main()
{
 static HURC                                  /* Menüfarben */
  m_mono =                  /* Menüfarben im Monochrom-Modus */
   {
    { INVERS, HNORMAL, NORMAL, INVERS, NORMAL },
    { INVERS, NORMAL, NORMAL, INVERS, UNDERLINE },
    INVERS                     /* Rahmenfarbe für Untermenüs */
   },
  m_color =                    /* Menüfarben für Color-Modus */
   {
    {                               /* Farben für Hauptmenü */
     COL( GELB, CYAN ), COL( WEISS, CYAN ),   /* schlummert */
     COL( GELB, BLAU), COL(WEISS, BLAU ),          /* aktiv */
     COL( SCHWARZ, CYAN )                       /* disabled */
    },
    {                              /* Farben für Untermenüs */
     COL( SCHWARZ, CYAN ), COL( WEISS, CYAN ),  /*schlummert*/
     COL( WEISS, BLAU ), COL( CYAN, BLAU ),         /* aktiv */
     COL( SCHWARZ, WEISS )                      /* disabled */
```

Listing 2: *(Fortsetzung)*

```
  },
  COL( WEISS, CYAN )          /* Rahmenfarbe für Untermenüs */
 };

static BEREICH felder[] =                          /* Mausfelder */
 {                   /* Textfenster, Slider und Sliderbuttons */
  {  1,  2, 78, 21, MPM( PTRSAMECHAR,       PTRINVCOL ) },
  { 79,  3, 79, 20, MPM( PTRDIFCHAR(Øx12), PTRINVCOL ) },
  {  2, 22, 77, 22, MPM( PTRDIFCHAR(Øx1d), PTRINVCOL ) },
  {  1, 22,  1, 22, MPM( PTRSAMECHAR,       PTRINVCOL ) },
  { 78, 22, 78, 22, MPM( PTRSAMECHAR,       PTRINVCOL ) },
  { 79, 21, 79, 21, MPM( PTRSAMECHAR,       PTRINVCOL ) },
  { 79,  2, 79,  2, MPM( PTRSAMECHAR,       PTRINVCOL ) },
 };

MEV ev;                                         /* Ereignismaske */
BYTE i,                                      /* Schleifenzähler */
     ende;     /* markiert das Ende der Programmausführung */

VioInit();                           /* Videomodul initialisieren */
KbmInit();          /* Tastatur- und Mausmodul initialisieren */

/*-- Demobildschirm aufbauen -----------------------------*/

VioClearScreen( WB );
VioFrame( VL( Ø), VO( 1), VR( Ø), VU( Ø), EINRA, WB );
VioFill(  VR( Ø), VO( 3), VR( Ø), VU(-4), '█', WS );
VioFill(  VL( 2), VU(-2), VR(-2), VU(-2), '█', WS );
VioPrint( VL( 1), VU(-2), INVERS, FALSE,  "\x1b" );
VioPrint( VR(-1), VU(-2), INVERS, FALSE,  "\x1a" );
VioPrint( VR( Ø), VO( 2), INVERS, FALSE,  "\x18" );
VioPrint( VR( Ø), VU(-3), INVERS, FALSE,  "\x19" );
VioPrint( VL(29), VO( 1), HRB, FALSE, " MENDEMO: untitled ");

/*-- Hilfszeilen als Demotext im Textfenster ausgeben -----*/
for (i = Ø; i < VioGetLines() - 5; ++i )
 VioPrint( VL(4), VO(2+i), WB, FALSE, cstr[i] );

VioClear( VL(Ø), VU(-1), VR(Ø), VU(-1), SV );
MSG( 1 , "MenGet() :" );
VioSetCursor( 1, 2 );                  /* Cursor in Textfenster */

/*-- Arbeit mit Pull-Down-Menü vorbereiten ----------------*/

MenSetCol( VioIsColor() ? m_color : m_mono );     /* Farben */
MenRegister( hauptmenue, ELVEK(hauptmenue), 1 );
MenDefToggle( MEN_EDIT+7, TRUE );  /* Read-Only ist Toggle */
MenDefToggle( MEN_OPTIONSNF+2, FALSE );  /* Full-Menu auch */
MenDisableMen( MEN_SEARCH+6 );     /* Function ist disabled */
MenSetFkt( tfkt );    /* TFKT wird bei Menüwechsel aufger. */


/*-- Koordinaten der Mausfelder setzen --------------------*/

felder[Ø].y2 = felder[5].y1 = felder[5].y2 = VioGetLines()-4;
felder[2].y1 = felder[2].y2 = felder[3].y1 =
felder[3].y2 = felder[4].y1 = felder[4].y2 = VioGetLines()-3;
felder[1].y2 = VioGetLines() - 5;
MenSetFelder( ELVEK(felder), felder );

/*-- Farben für den Mauscursor in den Mausfeldern setzen --*/

if ( VioIsColor() )                         /* Color-Modus? */
 {               /* Ja, Mauscursor für die Bereiche setzen */
  felder[Ø].ptr_mask = felder[3].ptr_mask =
  felder[4].ptr_mask = felder[5].ptr_mask =
  felder[6].ptr_mask = MPM( PTRSAMECHAR, MCOL );
  felder[1].ptr_mask = MPM( PTRDIFCHAR(Øx12), MCOL );
  felder[2].ptr_mask = MPM( PTRDIFCHAR(Øx1d), MCOL );
  MouSetDefaultPtr( MPM( PTRSAMECHAR, MCOL ) );
 }
else                                    /* im Monochrom-Modus */
 MouSetDefaultPtr( MPM( PTRSAMECHAR, PTRINVCOL ) );

MenPrintMen();                          /* Menüleiste aufbauen */
MouShowMouse();                         /* Mauscursor anzeigen */

ende = FALSE;           /* Demoprogramm noch nicht beenden */
```

Listing 2: *(Fortsetzung)*

MD( ) isoliert Sie als Anwender des MEN-Moduls also vom Aufbau der HMDES-Struktur mit dem Ziel, eine gewisse Übersichtlichkeit innerhalb Ihres Programmlistings zu gewährleisten.

Neben dem Namen des jeweiligen Menüs (bzw. des Variablennamens, unter dem das Menü definiert wurde) muß MD( ) auch die Nummer des Menüs übergeben werden, die später zur Identifikation des ausgewählten Menüs dient. Bei dieser Nummer kann es sich um eine beliebige Zahl zwischen 0 und 255 handeln, die die Kennummer des jeweiligen Hauptmenüs darstellt. Von dieser Nummer ausgehend werden die einzelnen Untermenüs fortlaufend numeriert, so daß das erste Untermenü die Nummer x+1, das zweite die Nummer x+2 usw. trägt. Beachten Sie dabei bitte, daß in dieser Rechnung auch MEN_TRENNER mitgerechnet werden, da sie als ganz normale Untermenüs gelten.

Um Überschneidungen der Kennummern zwischen den einzelnen Menüs zu vermeiden, sollte der Abstand zwischen den einzelnen Kennummern immer mindestens die Anzahl der Untermenüs plus 1 betragen. Wenn also beispielsweise das erste Menü die Kennnumer 0 trägt und über vier Untermenüs verfügt, dann darf das darauffolgende Menü erst mit der Kennummer 5 beginnen. Da in der Regel weit weniger als 255 Menüs und Untermenüs definiert werden, kann man diesem Problem jedoch einfach aus dem Weg gehen, indem man die Kennummer jeweils um einen konstanten Wert inkrementiert, wie es das obige Beispiel demonstriert.

Die Variable hauptmenue[] in dem obigen Beispiel führt uns zurück zur Funktion MenRegister( ), denn eine Variable von diesem Typ muß dieser Funktion als erster Parameter übergeben werden. Der zweite Parameter gibt die Anzahl der Hauptmenüs und damit der Einträge in dem übergebenen Vektor an, während der dritte Parameter die Anzahl der Hauptmenüs repräsentiert, die auf der rechten Seite der Menüleiste aufgeführt werden sollen. Im obigen Beispiel betrüge dieser Wert 1, da wir bei der Beschreibung des CUA-Standards gesehen haben, daß das »Hilfe«-Menü immer auf der rechten Seite der Menüleiste erscheinen sollte.

Dem Aufruf der Funktion MenRegister() können Aufrufe der Funktionen MenDisableMen( ) und MenSetToggleOn( ) bzw. MenSetToggleOff( ) folgen. Allen drei Funktionen ist gemein, daß ihnen die Kennummer des jeweiligen Haupt- oder Untermenüs übergeben werden muß, auf das sie sich beziehen.

MenDisableMen( ) legt fest, daß ein Haupt- oder Untermenü zum jetzigen Zeitpunkt nicht aufgerufen werden kann. Dies hat zur Folge, daß das Menü farblich hervorgehoben wird und es mit Hilfe der Cursortasten nicht als das aktuelle Menü markiert werden kann. Außerdem wechselt der Mauscursor sein Erscheinungsbild, sobald er sich über einem solchen Menü befindet und zeigt dann ein großes »X« an.

Sinn macht die Verwendung dieser Funktion nur dann, wenn ein Menü erst in einem bestimmten Kontext, z.B. nach der Ausführung eines anderen Befehls aufgerufen werden kann, weil erst dann die benötigten Informationen zu seiner Ausführung bereitstehen. Sie können das Menü daher zunächst mit Hilfe der Funktion `MenDisableMen()` deaktivieren, um seine Auswahl später über die Funktion `MenEnableMen()` wieder zu erlauben.

`MenSetToggleOn()` und das verwandte `MenSetToggleOff()` dienen zur Definition eines Untermenüs als Schalter, wobei jeweils die Kennummer des Untermenüs als Parameter beim Funktionsaufruf angegeben werden muß. Bei `MenSetToggleOff()` steht der Schalter zunächst auf Aus, während er bei `MenSetToggleOn()` auf An steht. Mit jedem Aufruf des Untermenüs wechselt dabei die Stellung des Schalters, so daß er jeweils abwechselnd auf An oder Aus steht. Mit Hilfe der beiden Funktionen können Sie seine Stellung jedoch jederzeit in Ihrem Sinne festlegen.

Während Sie sich dieser vier Funktionen nur bedienen müssen, wenn Sie mit Schaltern arbeiten oder ein Menü deaktivieren möchten, muß die Funktion `MenSetCol()` nach dem Aufruf von `MenRegister()` auf jeden Fall aufgerufen werden. Ihre Aufgabe ist es, der Menüverwaltung die Farben der einzelnen Hauptmenüs, der Hotkeys, sowie der Untermenüs und der Rahmen um die Untermenüs mitzuteilen. Die Verwendung dieser Funktion demonstriert Ihnen das Demoprogramm `MENDEMO.C` (Listing 4).

Eine weitere Funktion, derer Sie sich nicht bedienen müssen, die Sie aber bei der Erstellung der Benutzeroberfläche unterstützen kann, ist die Funktion `MenSetFelder()`. Mit Ihrer Hilfe können Sie verschiedene Mausbereiche innerhalb des Arbeitsbereichs der Applikation definieren, denen eine besondere Bedeutung innewohnt und die die Ausführung einer bestimmten Aktion zur Folge haben, sobald der linke Mausknopf niedergedrückt wird und sich der Mauscursor über einem dieser Felder befindet. Die Syntax dieser Funktion ist mit der der Funktion `MouSetBereich()` identisch, die in der letzten Folge dieser Serie (Thema Maus und Tastatur) vorgestellt wurde. Das Demoprogramm verwendet diese Funktion z.B., um die Bildschirmbereiche zu definieren, in denen sich eine horizontale bzw. eine vertikale Slider-Box (eine Art Scroll-Balken) befindet. Wie die Applikation feststellen kann, ob einer dieser Bereiche durch Betätigung des linken Mausknopfs angewählt wurde, werden wir noch sehen.

Nicht unerwähnt bleiben sollte an dieser Stelle auch die Funktion `MenSetFkt()`, mit deren Hilfe eine Funktion definiert werden kann, die immer dann aufgerufen wird, wenn ein anderes Haupt- oder Untermenü zum aktuellen Menü gewählt wird, wobei der angegebenen Funktion die Nummer des neuen aktuellen Menüs übergeben wird. Ein besonderer Fall ist dabei die Konstante `KEIN_MENU`, die übergeben wird, sobald das Menü durch die Betätigung von [Esc] oder die Auswahl eines Menüs bzw. Untermenüs verlassen wird.

```
do
 {
  ev = MenGet();                                    /* Ereignis holen */
  VioClear( VL(12), VU(-1), VR(0), VU(-1), SV );

  if ( ev.code == MEV_TASTE )                      /* Taste betätigt? */
   {                                                            /* Ja */
    switch( ev.d.key )                           /* Befehlsemulation */
     {
      case ALT_F4 :                                           /* Exit */
        ev.code = MEV_BEFEHL;          /* Taste löst Befehl aus */
        ev.d.befehl = MEN_FILE+12;                /* Befehlsnummer */
        break;

      case F2 :                                     /* Open Last File */
        ev.code = MEV_BEFEHL;          /* Taste löst Befehl aus */
        ev.d.befehl = MEN_FILE+3;                 /* Befehlsnummer */
        break;

      /*-- in dieser Art und Weise können auch alle ------*/
      /*-- anderen Tasten Befehlen zugeordnet werden ------*/

      case F1 :                 /* Aufruf des Helpmenüs emulieren */
        KbdUngetKey( ALT_H );        /* ALT+H in den Tastaturp. */
        break;

      default :                                /* jede andere Taste */
       MSG( 12, "Taste betätigt" );
       if ( IsAKey( ev.d.key ) )                  /* Ascii-Zeichen? */
        MSGF( 27, "(%c)", (char) ev.d.key );
       else                            /* erweiterter Tastaturcode */
        MSGF( 27, "(erweiterter Tastaturcode %d)", ev.d.key);
     }
   }


  if ( ev.code == MEV_MAUS )        /* Mausknopf über Bereich? */
   {                             /* Ja, Art des Bereichs auswerten */
    MSG( 12, "Mausklick über " );
    switch( ev.d.bereich )                 /* Bereich untersuchen */
     {
      case 0 : MSG( 27, "Textfenster" );
               VioSetCursor( MouGetCol(), MouGetRow() );
               break;
      case 1 : MSG( 27, "vertikaler Slider-Box" );
               break;
      case 2 : MSG( 27, "horizontaler Slider-Box" );
               break;
      case 3 : MSG( 27, "Pfeil nach links" );
               break;
      case 4 : MSG( 27, "Pfeil nach rechts" );
               break;
      case 5 : MSG( 27, "Pfeil nach unten" );
               break;
      case 6 : MSG( 27, "Pfeil nach oben" );
               break;
     }
    KbmEventWait( EV_LEFT_REL );      /* auf Loslassen warten */
   }


  if ( ev.code == MEV_BEFEHL )              /* Befehl angewählt? */
   {                                                            /* Ja */
    MSGF( 12, "Befehl mit der Nummer %d ausgewählt",
          ev.d.befehl );
    switch ( ev.d.befehl )                    /* Befehl auswerten */
     {
      case MEN_FILE+12 :                                      /* Exit */
        ende = TRUE;                      /* Demoprogramm beenden */
        break;

      case MEN_EDIT+7 :                                  /* Read-Only */
        break;

      case MEN_OPTIONSNF+2 :                        /* Full-Menus an */
        MenReplaceSubMen( MEN_OPTIONSNF, &of );
        MenDefToggle( MEN_OPTIONSF+5, TRUE );
        break;
```

Listing 2: *(Fortsetzung)*

```
        case MEN_OPTIONSF+5 :                   /* Full-Menus aus */
          MenReplaceSubMen( MEN_OPTIONSF, &onf );
          MenDefToggle( MEN_OPTIONSNF+2, FALSE );
          break;

        default :                               /* jeder andere Befehl */
          NA();                       /* Befehl ist nicht implementiert */
          break;
      }
    }
  }
  while ( !ende );                         /* wiederholen, bis EXIT */

  MouHideMouse();                          /* Mauscursor ausblenden */
  VioClearScreen( WB );                      /* Bildschirm löschen */
  VioSetCursor( 0, 0 );                /* Cursor in obere linke Ecke */
}
```

Listing 2: *(Ende)*

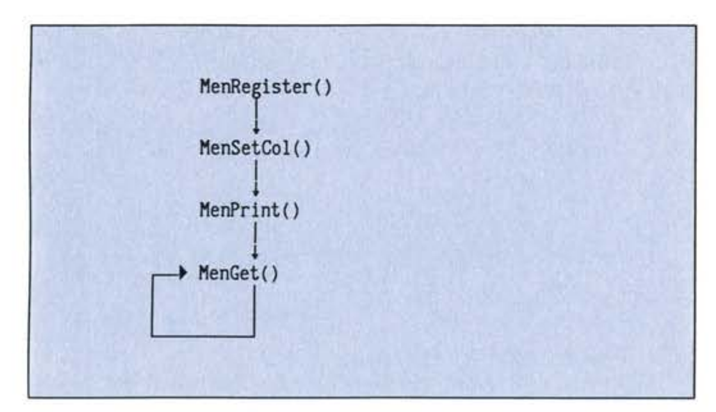


Bild 2: *Funktionsaufrufe zur Arbeit mit dem MEN-Modul.*

Die letzte Funktion, die vor dem Beginn der eigentlichen Menüabfrage aufgerufen werden muß, ist die Funktion `MenPrintMen()`. Sie baut die Menüleiste auf und bereitet den Bildschirm damit auf die Menüauswahl vor. Dies erfolgt über den Befehl `MenGet()`, der innerhalb einer Schleife permanent aufgerufen werden sollte, bis der Anwender den Befehl gibt, die Ausführung des Programms zu beenden. `MenGet()` geht jeweils davon aus, daß der Anwender sich noch nicht im Hauptmenü befindet und kehrt erst dann zum Aufrufer zurück, wenn eines der drei folgenden Ereignisse eingetreten ist:

- der Anwender befand sich noch nicht im Hauptmenü und betätigte eine Taste, die nicht zur Aktivierung des Hauptmenüs oder Auswahl einer seiner Menüs führte (`MEV_TASTE`);
- der Anwender hat ein Untermenü innerhalb eines Hauptmenüs oder ein Hauptmenü ausgewählt, das nicht über Untermenüs verfügt (`MEV_BEFEHL`);
- der Anwender hat den linken Mausknopf niedergedrückt, während sich der Mauscursor über einem der Bereiche befand, die über den Befehl `MenSetFelder()` definiert wurden (`MEV_MAUS`).

Welches dieser Ereignisse eingetreten ist, erkennen Sie anhand des Rückgabewertes der Funktion `MenGet()`, einer Struktur vom Typ `MEV`. Das Feld `CODE` enthält eine der oben aufgeführten Konstanten `MEV_...` und spiegelt damit die Art des Ereignisses wieder. Detaillierte Informationen im Zusammenhang mit dem Ereignis enthält die Variante `D` innerhalb der zurückgelieferten Struktur. Im Fall von `MEV_TASTE` enthält das Feld `KEY` den Tastencode der betätigten Taste, für `MEV_MAUS` enthält das Feld `BEREICH` die Nummer des Mausbereichs (erster Bereich = 0) und nach der Auswahl eines Untermenüs (`MEV_BEFEHL`) enthält das Feld `BEFEHL` die Kennummer dieses Befehls.

Wie auf das jeweilige Ereignis eingegangen und die Schleife zum wiederholten Aufruf von `MenGet()` aufgebaut werden kann, verdeutlicht wiederum das Demoprogramm `MENDEMO.C`. Auch die sinnvolle Verwendung von `MenSetFkt()` macht dieses Programm deutlich, indem es diese Funktion zur Installation einer Funktion nutzt, die jeweils Hilfeinformationen zu dem jeweils aktuellen Haupt- bzw. Untermenü in der untersten Bildschirmzeile ausgibt.

Über die bisher vorgestellten Funktionen hinaus hat das `MEN`-Modul noch einige andere Funktionen zu bieten, die sich in verschiedenen Situationen als sehr nützlich erweisen können. Die Funktion `MenReplaceMen()` beispielsweise erlaubt es Ihnen, einem Untermenü während der Arbeit mit dem `MEN`-Modul einen neuen Namen zu geben. Dies darf allerdings nicht in dem Sinne verstanden werden, daß ein Untermenü einen komplett neuen Namen erhält, denn dies würde sicherlich nicht gerade zur erstrebten Einheitlichkeit der Benutzeroberfläche beitragen. Vielmehr sollte diese Funktion in Verbindung mit Untermenüs angewendet werden, deren Ausführung sich auf bestimmte Objekte bezieht, wobei diese Objekte innerhalb des Menünamens genannt werden.

In einer Datenbankapplikation wird es z.B. einen Befehl mit dem Namen »Öffnen« geben, der die Arbeit mit einer bestimmten Datenbank einleitet. Andere Befehle wie z.B. der »Reorganisations«-Befehl beziehen sich auf die vom Anwender im Verbindung mit dem »Öffnen«-Befehl spezifizierte Datenbank, und so erscheint es durchaus sinnvoll, den Namen der Datenbank nach Aufruf des »Öffnen«-Befehls mit Hilfe der Funktion `MenReplaceName()` an das Ende des »Reorganisations«-Befehls anzuhängen und dem Anwender dadurch deutlich zu machen, auf welche Datenbank sich der oder die Befehle beziehen.

Der Aufruf dieser Funktion gestaltet sich dabei recht simpel, da lediglich die Kennummer des angesprochenen Untermenüs und der neue Menüname übergeben werden müssen. Dabei sind jedoch zwei wichtige Regeln zu beachten:

- die Breite des Untermenüs darf sich durch den Aufruf der Funktion nicht verändern (das Untermenü darf nicht breiter sein als das bisher breiteste Untermenü) und
- das neue Menü muß über den selben Hotkey verfügen, wie das Menü, das es ersetzt.

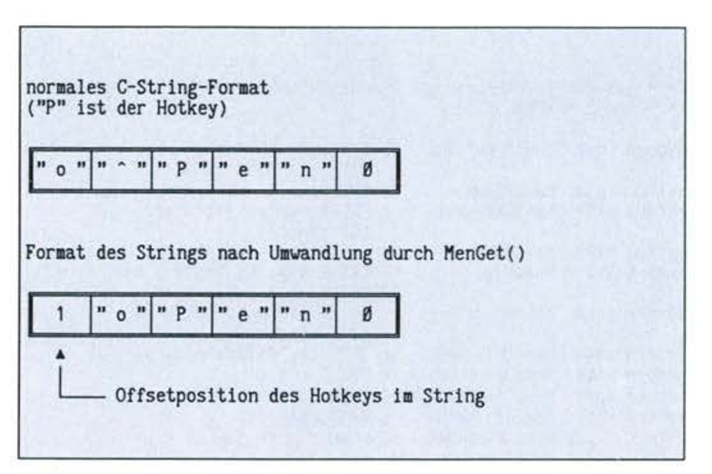


Bild 3: *Kodierung eines Menünamens durch MenGet()*

Sollten Sie den Namen des neuen Menüs innerhalb des Puffers aufbauen, der auch den vorhergehenden Menünamen enthielt, so beachten Sie bitte, daß das MEN-Modul die verschiedenen Namen-Strings beim Aufruf der MenRegister()-Funktion »umbaut«. Um nämlich die Position des Hotkeys innerhalb des Menüs nicht immer wieder suchen zu müssen, wird der String so umgebaut, daß das erste Byte im String die Offsetposition des Hotkeys innerhalb des Strings aufnimmt, während gleichzeitig das »^«-Zeichen aus dem String gelöscht wird, damit der vorgegebene Speicherplatz für die Darstellung des Strings weiterhin ausreicht. Das führt natürlich dann zu Problemen, wenn das Menü nicht über einen Hotkey verfügt, also das »^«-Zeichen im String fehlt und damit kein Platz für ein weiteres Byte vorhanden ist, das die Offsetposition des (nicht vorhandenen) Hotkeys angeben könnte.

In diesem Fall behilft sich das MEN-Modul, indem es den String so läßt wie er ist. Geht man davon aus, daß sich der Hotkey auf jeden Fall unter den ersten 32 Zeichen des Menünamens befindet und nimmt man weiterhin an, daß kein Menü mit einem Zeichen eingeleitet wird, dessen ASCII-Code kleiner 32 ist, so kann das MEN-Modul anhand des ersten Byte eines Menünamens jedoch jederzeit erkennen, ob das Menü über einen Hotkey verfügt, oder nicht.

Damit Sie bei der nachträglichen Manipulation von Menünamen nicht mit dieser Art der Kodierung in Konflikt geraten, können Sie sich der Funktion MenConvNameBack( ) bedienen, der lediglich ein Pointer auf den adressierten Menünamen übergeben werden muß. Sie wandelt diesen Namen in sein ursprüngliches Format zurück, so daß Sie ihn ohne Schwierigkeiten mit Hilfe der String-Funktionen aus der C-Bibliothek bearbeiten können.

Nicht nur den Namen eines einzelnen Menüs, sondern ein ganzes Hauptmenü samt seines Umtermenüs können Sie während der Programmausführung mit Hilfe der Funktion MenReplaceSubMen( ) austauschen. Eine mögliche Anwendung dieser Funktion macht das Demoprogramm MENDEMO.C deutlich. Das »Option«-Menü existiert in diesem Programm in zwei Versionen: einer normalen, mit nur wenigen Befehlen, die für den Programm-Einsteiger gedacht ist, und einer, die einige zusätzliche Befehle enthält und sich damit an die Anwender richtet, die mit dem Programm bereits intensiver vertraut sind. Beide Menüs enthalten zur Umschaltung einen Menüpunkt mit dem Namen »Full Menus« der jeweils zwischen den beiden Darstellungsweisen hin- und herschaltet.

Einzige Bedingung für die Arbeit mit der Funktion MenReplaceSubMen( ) ist dabei, daß sich der Name des Hauptmenüs und dessen Hotkey durch den Aufruf dieser Funktion nicht ändert, während die Untermenüs komplett ausgetauscht werden können.

Nach dieser Übersicht über die wichtigsten Funktionen, die Ihnen des MEN-Modul zur Arbeit mit SAA-Menüs zur Verfügung stellt, hier noch ein Kurzüberblick über die weiteren Funktionen:

MenSetSchattenOn( ), MenSetSchattenOff( )

Diese Funktionen schalten den Schatten unter den Untermenü-Fenstern an oder aus und können jederzeit aufgerufen werden.

MenSetRahmen( )

definiert die Art des Rahmens, der um ein Untermenü-Fenster gezogen wird. Der horizontale und der vertikale Rahmen können individuell eingestellt werden, wobei jeweils eine der Konstanten EINRA oder DOPRA übergeben werden muß.

MenSetHMKey( ), MenGetHMKey( )

erlauben die Definition bzw. die Abfrage der Taste, über die das Hauptmenü aktiviert wird. Standardmäßig ist die Taste [F10] (SAA-Standard) voreingestellt, doch kann mit Hilfe von MenSetHMKey( ) auch jede andere Taste definiert werden.

MenWinOpen( ), MenWinClose( )

Diese Funktionen, die innerhalb des MEN-Moduls selbst zum Öffnen und Schließen der Untermenü-Fenster aufgerufen werden, können Sie auch in Ihren Applikationen aufrufen, wenn es darum geht, Fenster zu öffnen, sie automatisch mit einem Rahmen und mit einem Schatten zu versehen.

Integration des MEN-Moduls

Um das MEN-Modul in Ihren Programmen einsetzen zu können, müssen Sie neben den hier vorgestellten Modulen MEN.C und MENUTIL.C auch über die Module VIO.C und KBM.C verfügen, die in den beiden vorhergehenden Folgen dieser Serie vorgestellt wurden. Die Include-Dateien dieser drei Module müssen Sie über den #INCLUDE-Befehl in ihre Applikation einbinden. Die Module müssen unter einem der verschiedenen Speichermodelle des Microsoft C-Compilers kompiliert und die entstandenen Object-Dateien dann mit Ihrem Programm innerhalb des Linkvorgangs verbunden werden. Ein Beispiel für diesen Vorgang gibt die fol-

genden Befehlszeile, die den Microsoft C-Compiler aufruft, um das Demoprogramm `MENDEMO.C` unter dem Speichermodell `SMALL` zu entwickeln.

```
CL /AS mendemo.c men.c menutil.c kbm.c vio.c
```

Michael Tischer

```
/*************************************************************/
/*                        M E N . C                          */
/*-----------------------------------------------------------*/
/*    Aufgabe        : Stellt verschiedene Funktionen zur    */
/*                     Arbeit mit Pull-Down-Menüs bereit.    */
/*-----------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                       */
/*    entwickelt am  : 22.02.1989                            */
/*    letztes Update : 14.03.1989                            */
/*-----------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] MEN.C MENUTIL.C /C   */
/*                     dann mit einem anderen Modul linken   */
/*************************************************************/

#define SAA_MEN_INTERN            /* Men-Modul wird kompiliert */

/*== Include-Dateien einbinden ==============================*/

#include "vio.h"          /* Zugriff auf die Video-Funktionen */
#include "kbm.h"              /* Zugriff auf Tastatur & Maus */
#include "men.h"           /* eigene Include-Datei einbinden */
#include <stdlib.h>
#include <memory.h>
#include <malloc.h>
#include <string.h>

/*== globale Variablen ======================================*/

TASTE   hmkey = (TASTE) MEN_HM_KEY;  /* aktiviert Hauptmenü */
BYTE    hmenanz,            /* Anzahl der Einträge im Hauptmenü */
        hmenrechts,              /* Anzahl der Einträge rechts */
        fanz,                    /* Anzahl weiterer Mausfelder */
        aktmen,             /* die Nummer des aktuellen Menüs */
        schatten = MEN_SCHATTEN,      /* Schatten um Menü-F. */
        horat = MEN_HORAT,       /* horizontaler Menürahmen */
        verat = MEN_VERAT;          /* vertikaler Menürahmen */
HMDES   *hmenptr;               /* Ptr auf den Hauptmenü-Vektor */
MENFKT  menfkt = (MENFKT) 0;        /* Aufruf bei Menüwechsel */

/*-- Pointer auf Vektoren mit Bereichsbeschreiben ----------*/
BEREICH *berhm = (BEREICH *) 0,                   /* Hauptmenü */
        *fptr,                         /* weitere Mausbereiche */
        *berall,          /* Vektor verknüpft berhm und fptr */
        *aptr;  /*Mausber., Hauptmenü und das akt. Untermenü*/

/*-- Pointer auf Vektoren mit Hotkeys für das Hauptmenü ----*/
TASTE   *hmaltkey,                                /* ALT-Codes */
        *hmnorkey;                         /* Kleinbuchstaben */

HURC    farben;    /* Farben für Hauptmenü, Untermenüs etc. */
```

Listing 3: *MEN.C*

```
/*== externe Referenzen auf die Hilfsfunktionen aus dem ====*/
/*== Modul MENUTIL.C                                    ====*/

extern void  MenISchatten    ( BYTE x1, BYTE y1,
                               BYTE x2, BYTE y2 );
extern void  MenINewMen      ( BYTE nr );
extern BYTE  MenISmSearch    ( HMDES *mptr, int start,
                               int step );
extern BYTE  MenIHmSearch    ( int start, int step );
extern int   MenIKeyIn       ( TASTE key, RG TASTE * buf,
                               BYTE anz );
extern void  MenIColorSMen   ( RG HMDES *mptr, BYTE nr,
                               BYTE choice );
extern void  MenIColorHMen   ( BYTE nr, BYTE choice );
extern void  MenIOpenSubMen  ( BYTE nr );
extern void  MenICloseSubMen ( void );
extern HMDES *MenIFindMen    ( BYTE menr );
extern TASTE MenIConvName    ( char * nptr );
extern void  MenIFreeMem     ( void );
extern BYTE  MenIGetBereich  ( BYTE * ptr );

/************************************************************
*  Funktion         : M e n S e t F e l d e r               *
**--------------------------------------------------------**
*  Aufgabe          : Registriert weitere Mausfelder.       *
*  Eingabe-Parameter: ANZ  = Anzahl der Felder              *
*                     BPTR = Pointer auf den Vektor mit den *
*                            Bereichsbeschreibern (BEREICH) *
*  Return-Wert      : keiner                                *
*  Info             : - Diese Funktion darf erst nach dem   *
*                       Aufruf von MenRegister() aufgerufen *
*                       werden.                             *
*                     - Der Vektor, auf den der übergebene  *
*                       Pointer zeigt, darf bis zum erneuten *
*                       Aufruf dieser Funktion nicht ver-   *
*                       ändert werden.                      *
************************************************************/

void MenSetFelder( BYTE anz, BEREICH *bptr )
{
 fptr = bptr;          /* Pointer in globaler Var. speichern */
 fanz = anz;                               /* Anzahl ebenfalls */

 free( (void *) berall );   /* gemeinsamen Vektor freigeben */

 /*-- den Vektor BERALL neu aufbauen ----------------------*/

 berall = (BEREICH *) malloc( ( hmenanz + fanz ) * SB );
 memcpy( berall, fptr, fanz * SB );
 memcpy( berall + fanz, berhm, hmenanz * SB );
}

/************************************************************
*  Funktion         : M e n S e t F l a g                   *
**--------------------------------------------------------**
*  Aufgabe          : Setzt das Toggle- oder das Disable-   *
*                     Flag eines Menüs.                     *
*  Eingabe-Parameter: MENR = Nummer des Untermenüs          *
*                     FLAG = TRUE für Toggle-Flag, FALSE für *
*                            DISABLE-Flag                   *
*                     WERT = TRUE für ON, FALSE für OFF     *
*  Return-Wert      : keiner                                *
************************************************************/

void MenSetFlag( BYTE menr, BYTE flag, BYTE wert )
{
 RG ULONG *flagptr;                   /* Pointer auf das Flag */
 HMDES    *hptr;              /* Ptr auf den HMDES des Menüs */
 BYTE     relnr;                    /* realtive Menünummer */
 PTRVIEW  moucur;                        /* neuer Mauscursor */

 if ( hptr = MenIFindMen( menr ) )               /* Menü suchen */
  {                                 /* das Menü wurde gefunden */
   relnr = menr - hptr->knr;  /* relative Menünr. berechnen */

   /*-- Pointer auf zu manipulierendes Flag setzen ---------*/

   flagptr = flag ? &(hptr->tontoff) : &(hptr->disable);
```

Listing 3: *(Fortsetzung)*

```
    if ( wert )                           /* Toggle anschalten? */
      *flagptr |= ( 1 << relnr );             /* Ja, Bit setzen */
    else                                              /* Nein */
      *flagptr &= ~( 1 << relnr );               /* Bit löschen */

    /*-- eine Veränderung des Disable-Flag hat auch die -----*/
    /*-- Veränderung des Maus-Cursors für dieses Menü   -----*/
    /*-- zur Folge.                                     -----*/

    if ( flag == FALSE )              /* Disable-Flag gesetzt? */
     {         /* Ja, Maus-Cursor für den Bereich neu setzen */
      moucur = wert ? MEN_DIS_CUR : MEN_STD_CUR;
      if ( relnr )         /* wurde ein Untermenü beeinflußt? */
       (hptr->smber+relnr-1)->ptr_mask = moucur;         /* Ja */
      else                 /* Nein, Menü ist ein Hauptmenü */
       {
        ( berhm+(hptr-hmenptr) )->ptr_mask = moucur;
        ( berall + fanz + (hptr-hmenptr) )->ptr_mask = moucur;
       }
     }
  }
}

/************************************************************
*  Funktion          : M e n D e f T o g g l e              *
**--------------------------------------------------------**
*  Aufgabe           : Definiert ein Untermenü als einen    *
*                      Schalter. (Toggle-Menü)              *
*  Eingabe-Parameter: MENR = Nummer des Untermenüs          *
*                      INIT = Initialisierungswert          *
*                             ( TRUE / FALSE )              *
*  Return-Wert       : keiner                               *
************************************************************/

void MenDefToggle( BYTE menr, BYTE init )
{
 RG HMDES *hptr;                 /* Ptr auf den HMDES des Menüs */
 BYTE     relnr;                     /* realtive Menünummer */

 if ( hptr = MenIFindMen( menr ) )           /* Menü suchen */
  {                               /* das Menü wurde gefunden */
   relnr = menr - hptr->knr;  /* relative Menünr. berechnen */
   hptr->toggles |= ( 1 << relnr );      /* Toggle-Bit setzen */
   if ( init )                        /* Toggle anschalten? */
    hptr->tontoff |= ( 1 << relnr );        /* Ja, Bit setzen */
  }
}

/************************************************************
*  Funktion          : M e n P r i n t M e n                *
**--------------------------------------------------------**
*  Aufgabe           : Gibt die Hauptmenüzeile aus.         *
*  Eingabe-Parameter: keine                                 *
*  Return-Wert       : keiner                               *
*  Info              : - Bei der Ausführung dieser Funktion *
*                        wird davon ausgegangen, daß zuvor  *
*                        MenRegister() aufgerufen wurde, um *
*                        das Menü zu initialisieren.        *
************************************************************/

void MenPrintMen( void )
{
 RG BEREICH *lptr;               /* Laufzeiger in BERHM-Vektor */
 RG HMDES   *mptr;             /* Laufzeiger in HMENPTR-Vektor */
 BYTE        i,                             /* Schleifenzähler */
             pos;   /* Spalte des Hotkey in einem Menünamen */

 /*-- Menüzeile löschen ----------------------------------------*/
 VioClear( 0, MEN_ZEILE, VioGetCols()-1, MEN_ZEILE,
           farben.hm.sn );

#if MEN_STRICH          /* Strich neben Menü rechts ausgeben? */
  if ( hmenrechts )                    /* gibt es Menüs rechts? */
   {         /* Ja, Ptr auf zugehörigen BEREICH ermitteln */
    lptr = berhm + hmenanz - hmenrechts;
    VioPrint(lptr->x1-2, MEN_ZEILE, farben.hm.sn, FALSE, "|");
   }
#endif               /* Ende der konditionalen Kompilierung */
```

Listing 3: *(Fortsetzung)*

```
 /*-- die einzelnen Menüs von links nach rechts ausgeben ---*/
 for ( i = hmenanz, lptr = berhm, mptr = hmenptr;
       i;
       --i, ++lptr, ++mptr )
  {             /* erst Menü ausgeben, dann Hotkey einfärben */
   VioPrint(lptr->x1, MEN_ZEILE,
            ( mptr->disable & 1 ) ? farben.hm.disable
                                   : farben.hm.sn,
            FALSE, *(mptr->mena)+1);
   pos = lptr->x1 + (BYTE) **(mptr->mena); /* Hotkey-Spalte */
   if ( (mptr->disable & 1) == 0 )          /* Menü anwählbar? */
    VioColor( pos, MEN_ZEILE, pos, MEN_ZEILE, farben.hm.sh );
  }
 MouSetBereich( hmenanz + fanz, berall );  /*Mausber. setzen*/
}

/************************************************************
*  Funktion          : M e n W i n O p e n                  *
**--------------------------------------------------------**
*  Aufgabe           : Baut ein Menü-Fenster inklusive eines *
*                      Schatten auf.                        *
*  Eingabe-Parameter: X1, Y1 = obere linke Fensterecke      *
*                      X2, Y2 = untere rechte Fensterecke   *
*                      HORA   = Art des horizontalen Rahmens *
*                      VERA   = Art des vertikalen Rahmens  *
*                      RCOL   = Farbe für den Rahmen        *
*  Return-Wert       : keiner                               *
*  Info              : Für HORA und VERA kann entweder DOPRA *
*                      (doppelter Rahmen) oder EINRA (ein-  *
*                      facher Rahmen) angegeben werden. Bei *
*                      der Übergabe von KEIN_RA wird kein   *
*                      Rahmen um das Fenster gezogen.       *
************************************************************/

void MenWinOpen( BYTE x1, BYTE y1, BYTE x2, BYTE y2,
                 BYTE hora, BYTE vera, BYTE rcol )

{
 static RAHMEN rtypen[4] =                    /* Rahmentypen */
  {
   { "┌", "┐", "└", "┘", '─', '│' },              /* Typ 0 */
   { "╒", "╕", "╘", "╛", '═', '│' },              /* Typ 1 */
   { "╓", "╖", "╙", "╜", '─', '║' },              /* Typ 2 */
   { "╔", "╗", "╚", "╝", '═', '║' },              /* Typ 3 */
  };
 BYTE       typ,                               /* Rahmentyp */
            sx,                      /* Spalten für Schatten */
            sy;                       /* Zeilen für Schatten */
 VEL far    *bptr;           /* Pointer auf allokierten Puffer */
 RG RAHMEN *rptr;           /* Pointer auf die Rahmenzeichen */

 MouHideMouse();                     /* Mauscursor ausblenden */

 if ( schatten )                       /* Schatten aufbauen? */
  {
   sx = ( tline > 25 ) ? 1 : 2;     /* Anz. Schattenspalten */
   if ( x2 + sx >= tcol )  /* mit Schatten zuviele Spalten? */
    sx = tcol - x2 - 1;                               /* Ja */
   sy = ( y2 == tline-1 ) ? 0 : 1;         /* Schattenzeilen */
  }
 else                                         /* kein Schatten */
  sx = sy = 0;                /* Schattenspalten & -zeilen = 0 */

 VioWinOpen( x1, y1, x2+sx, y2+sy );          /* Fenster öffnen */

 if ( sy )                         /* Schatten unter dem Fenster? */
  {                                                     /* Ja */
   if ( VioIsColor() )             /* Farbdarstellung möglich? */
    MenISchatten(x1 + (( tcol > 80) ? 1 : 2), y2+1, x2, y2+1);
   else                         /* Nein, monochromer Bildschirm */
    VioFill( x1 + ( ( tcol > 80 ) ? 1 : 2 ), y2+1, x2, y2+1,
             '█', NORMAL );
  }

 if ( sx )                        /* Schatten links vom Fenster */
  {                                                     /* Ja */
   if ( VioIsColor() )             /* Farbdarstellung möglich? */
    MenISchatten( x2+1, y1+1, x2 + sx, y2+sy );
```

Listing 3: *(Fortsetzung)*

```
  else                    /* Nein, monochromer Bildschirm */
   VioFill( x2+1, y1+1, x2 + sx, y2+sy, '█', NORMAL );
 }

VioSetView( x1, y1, x2, y2 );          /* View-Bereich setzen */
VioClear(VL(1), VO(1), VR(-1), VU(-1), rcol );  /* löschen */

/*-- Rahmen um Fenster ziehen ----------------------------*/

if ( hora != KEINRA  &&  vera != KEINRA )         /* Rahmen? */
 {                                                   /* Ja */
  typ = ( hora == DOPRA ) ? 1 : Ø;
  if ( vera == DOPRA )          /* vertikale Linien doppelt? */
   typ += 2;                               /* Ja, Bit 1 setzen */

  rptr = &rtypen[ typ ];   /* Ptr auf Rahmenzeichen setzen */

  VioPrint( VL(Ø), VO(Ø), rcol, FALSE, rptr->ol );
  VioPrint( VR(Ø), VO(Ø), rcol, FALSE, rptr->or );
  VioPrint( VL(Ø), VU(Ø), rcol, FALSE, rptr->ul );
  VioPrint( VR(Ø), VU(Ø), rcol, FALSE, rptr->ur );

  VioFill( VL(Ø), VO(1), VL(Ø), VU(-1), rptr->verti, rcol );
  VioFill( VR(Ø), VO(1), VR(Ø), VU(-1), rptr->verti, rcol );
  VioFill( VL(1), VO(Ø), VR(-1), VO(Ø), rptr->hori, rcol );
  VioFill( VL(1), VU(Ø), VR(-1), VU(Ø), rptr->hori, rcol );
 }
 MouShowMouse();                /* Mauscursor wieder anzeigen */
}

/***************************************************************
*  Funktion          : M e n I S m D r a g                      *
**-----------------------------------------------------------**
*  Aufgabe           : Überwacht die Maus während der Aus-    *
*                      wahl eines Untermenüs im Drag-Modus.   *
*  Eingabe-Parameter: MENR    = Nummer des ausgewählten Haupt-*
*                               menüs (Ø = Menü links)         *
*                      AUFPTR = Pointer auf ein Byte, das den *
*                               Wert TRUE enthält, wenn das    *
*                               Untermenüfenster aufgebaut     *
*                               werden soll.                   *
*  Return-Wert       : Struktur mit Informationen über das    *
*                      eingetretene Ereignis                   *
***************************************************************/

static MEV MenISmDrag( BYTE menra, BYTE *aufptr )
{
 RG HMDES *mptr;               /* Ptr auf den Menübeschreiber */
 MEV      eg;           /* nimmt Ereignis-Code und Daten auf */
 int      event;                             /* Ereignismaske */
 BYTE     btypa,          /* aktueller Typ des Mausbereichs */
          btypn,               /* neuer Typ des Mausbereichs */
          bnra,            /* Nummer des aktuellen Bereichs */
          bnrn,               /* Nummer des neuen Bereichs */
          menrn,           /* Nummer des neuen Hauptmenüs */
          relescape,    /* ESCAPE bei Loslassen des Mauskn. */
          aufbau;           /* Untermenüfenster aufbauen? */
 int      snr;            /* Nummer des aktuellen Untermenüs */

 menrn = menra;        /* sofortigem Schleifenende vorbeugen */
 aufbau = *aufptr;                     /* Aufbau-Flag laden */
 relescape = !*aufptr;     /* bei Loslassen Eingabe beenden? */
 do
  {
   mptr = hmenptr + menra;     /* Ptr auf Menübeschr. setzen */
   btypa = MenIGetBereich( &bnra );  /* Mausber. & Nr holen */
   if ( btypa == BER_SMEN  &&         /* bereits in Untermenü? */
        SubMenIsOk( mptr, bnra ) )
    snr = bnra;                           /* Ja, Nummer merken */
   else                               /* nicht in einem Untermenü */
    {
     snr = -1;                     /* kein Untermenü angezeigt */
     MenINewMen( mptr->knr );           /* Befehlsnr. Hauptmenü */
    }

   if ( aufbau )                  /* Untermenüfenster aufbauen? */
    {                                                   /* Ja */
     MenIColorHMen( menra, TRUE );   /* Hauptmenü einfärben */
```

Listing 3: *(Fortsetzung)*

```
     if ( mptr->anzahl )                 /* gibt es Untermenüs? */
      MenIOpenSubMen( menra );              /* Ja, Fenster öffnen */
    }
   else                       /* Untermenüfenster nicht aufbauen */
    aufbau = TRUE;                    /* Fenster ab jetzt aufbauen */

   do                                      /* Maus-Abfrageschleife */
    {
     event = KbmEventWait( EV_LEFT_REL | EV_MOU_MOVE );
     if ( event & EV_MOU_MOVE )                    /* Maus bewegt? */
      {                                                      /* Ja */
       btypn = MenIGetBereich( &bnrn );       /* Bereich holen */
       if ( btypn!=btypa || bnrn!=bnra )   /* neuer Bereich? */
        {                              /* Ja, jetzt neuer Bereich */
         if ( snr != -1 ) /* wurde ein Untermenü angezeigt? */
          MenIColorSMen( mptr, snr, FALSE );               /* Ja */

         switch ( btypn )   /* Art des Bereichs untersuchen */
          {
           case BER_SMEN :                          /* Untermenü */
            if ( SubMenIsOk( mptr, bnrn ) )   /* Menü o.k.? */
             {                             /* Ja, Menü einfärben */
              MenIColorSMen( mptr, snr = bnrn, TRUE );
              relescape = FALSE;
             }
            else                          /* kein neues Untermenü */
             {
              MenINewMen( mptr->knr );
              snr = -1;                     /* kein Menü angezeigt */
             }
            break;

           case BER_HMEN :                          /* Hauptmenü!? */
            if ( bnrn!=menra && MenIsOk( bnrn ) )
             menrn = bnrn;          /* HMen o.k., Nummer merken */
            break;

           case BER_MAUS :
           case BER_NO   :
            MenINewMen( mptr->knr );
            break;
          }
         btypa = btypn;        /* Art und Nummer des aktuellen */
         bnra = bnrn;          /* Bereichs merken              */
        }
      }                                 /* Ende der Mausschleife */
    }
   while ( menrn==menra && (event & EV_LEFT_REL)==Ø );

   if ( menrn != menra )                      /* neues Hauptmenü? */
    {                                                        /* Ja */
     if ( mptr->anzahl )                 /* gibt es Untermenüs? */
      MenICloseSubMen( );              /* Ja, Fenster schließen */
     MenIColorHMen( menra, FALSE ); /* Hauptmenü ausblenden */
     menra = menrn;              /* Nummer des neuen Hauptmenüs */
     relescape = FALSE;           /* bei Loslassen kein ESCAPE */
    }
  }    /* warten, bis der linke Mausknopf losgelassen wird */
 while ( ( event & EV_LEFT_REL ) == Ø );

 /*-- der linke Mausknopf wurde losgelassen -----------------*/
 /*-- Bereich auswerten, über dem sich die Maus befand -----*/

 eg.code = MEV_ENDE;                     /* von ENDE ausgehen */
 btypn = MenIGetBereich( &bnrn );  /* Mausbereich ermitteln */
 switch ( btypn )      /* je nach Bereich untersch. reagieren */
  {
   case BER_HMEN :                                 /* Hauptmenü */
    if ( relescape )                         /* Eingabe beenden? */
     eg.code = MEV_ENDE;                                    /* Ja */
    else                                                  /* Nein */
     {
      if ( MenIsOk( bnrn ) )                     /* Menü enabled? */
       {                                                     /* Ja */
        if ( mptr->anzahl )                 /* gibt es Untermenüs */
         {                                                   /* Ja */
          eg.code = MEV_POINT;               /* in Point-Modus w. */
          eg.d.befehl = bnrn;                /* Menünummer merken */
```

Listing 3: *(Fortsetzung)*

```
        *aufptr = FALSE; /*Untermenüfenster nicht aufbauen*/
       }
      else                      /* es gibt keine Untermenüs */
       {                                 /* Befehl anwählen */
        eg.code = MEV_BEFEHL;
        eg.d.befehl = mptr->knr;
       }
     }
   }
  break;

 case BER_SMEN :                               /* Untermenü */
  if ( SubMenIsOk( mptr, bnrn ) )         /* Untermenü o.k.? */
   {                                                  /* Ja */
    eg.code = MEV_BEFEHL;               /* Befehl angewählt */
    eg.d.befehl = mptr->knr + bnrn;          /* Befehlsnr. */
    if ( mptr->toggles & ( 1 << bnrn ) )  /* Toggle-Menü? */
     mptr->tontoff ^= ( 1 << bnrn );      /* Ja, umschalten */
   }
  break;
 }

/*-- das alte Untermenüfenster löschen --------------------*/
if ( eg.code != MEV_POINT )           /* in Point-Modus w.? */
 {                                                  /* Nein */
  if ( mptr->anzahl )                  /* gibt es Untermenüs? */
   MenICloseSubMen( );              /* Ja, Fenster schließen */
  MenIColorHMen( menra, FALSE );    /* Hauptmenü ausblenden */
 }

return eg;                 /* Ereignis-Struktur zurückliefern */
}

/************************************************************
* Funktion          : M e n I S m P o i n t                *
**--------------------------------------------------------**
* Aufgabe           : Überwacht die Eingaben von Tastatur  *
*                     und Maus bei der Auswahl eines Unter- *
*                     menüs im Point-Modus.                *
* Eingabe-Parameter: MENR   = Nummer des ausgewählten Haupt-*
*                              menüs (Ø = Menü links)      *
*                     AUFPTR = Pointer auf ein Byte, das den *
*                              Wert TRUE enthält, wenn das  *
*                              Untermenüfenster aufgebaut   *
*                              werden soll.                 *
* Return-Wert       : Struktur mit Informationen über das  *
*                     eingetretene Ereignis                *
************************************************************/

static MEV MenISmPoint( BYTE menr, BYTE *aufptr )
{
 #define S( p1, p2, p3 )   MenISmSearch( p1, p2, p3 )
 #define M( p1, p2 )       MenIHmSearch( p1, p2 )

 MEV   eg;          /* nimmt Ereignis-Code und Daten auf */
 TASTE key;                 /* dient der Tastaturabfrage */
 int   event;                          /* Ereignismaske */
 HMDES *mptr;               /* Ptr auf den Menübeschreiber */
 BYTE  btyp,                              /* Bereichstyp */
       bnr,                   /* Bereichsnummer für Maus */
       nr,      /* Nummer des aktuellen Menüs im Untermenü */
       code,             /* zeigt an, was weiter geschieht */
       newmenr,            /* Nummer des neuen Hauptmenüs */
       aufbau;              /* Untermenüfenster aufbauen? */
 int   new;                             /* neue Menünummer */
 aufbau = *aufptr;                    /* Aufbau-Flag laden */
 do
  {
   mptr = hmenptr + menr; /* Ptr auf Menübeschreiber setzen */

   if ( aufbau )       /* ist das Fenster bereits aufgebaut? */
    {                                               /* Nein */
     MenIColorHMen( menr, TRUE );      /* Hauptmenü einfärben */
     if ( mptr->anzahl )              /* gibt es Untermenüs? */
      MenIOpenSubMen( menr );           /* Ja, Fenster öffnen */
    }
   else                         /* Fenster ist bereits aufgebaut */
    aufbau = TRUE;              /* Fenster von jetzt an aufbauen */

   nr = MenISmSearch( mptr, Ø, +1 );  /* erstes Menü suchen */
   MenIColorSMen( mptr, nr, TRUE ); /*Menü als gewählt mark.*/
   eg.code = MEV_NO;                 /* noch nichts passiert */

   do                                      /* Eingabeschleife */
    {         /* auf Ereignis von Maus oder Tastatur warten */
     event = KbmEventWait( EV_KEY_AVAIL | EV_LEFT_PRESS );
     if ( event & EV_KEY_AVAIL )          /* Taste betätigt? */
      {                                                 /* Ja */
       eg.code = MEV_MOVE;         /* von neuem Menü ausgehen */

       switch ( key = KbdGetKey() )                     /* Ja */
        {
         case ESC    :                 /* Menüauswahl beenden */
          eg.code = MEV_ENDE;
          break;

         case CR     :            /* aktuelles Menü ausgewählt */
           eg.code = MEV_BEFEHL;
           new = nr;
           break;

         case CHOME  :                /* erstes Menü auswählen */
           new = S( mptr, Ø, +1 );
           break;

         case CEND   :                /* letztes Menü auswählen */
           new = S( mptr, mptr->anzahl+1, -1 );
           break;

         case SPACE :      /* das nachfolgenden Menü anwählen */
         case CDOWN :
           new = S( mptr, nr, +1);
           break;

         case BS     :       /* das vorhergehende Menü anwählen */
         case CUP    :
           new = S( mptr, nr, -1);
           break;

         case CLEFT :       /* in das Hauptmenü links wechseln */
           eg.code = MEV_HAUPTMEN;           /* neues Hauptmenü */
           newmenr = M( menr, -1 );
           break;

         case CRIGHT:      /* in das Hauptmenü rechts wechseln */
           eg.code = MEV_HAUPTMEN;           /* neues Hauptmenü */
           newmenr = M( menr, +1 );
           break;

         default     :                           /* Buchstabe? */
           new = MenIKeyIn( key, mptr->awb, mptr->anzahl );
           if (new!=-1 && (mptr->disable & (1 << ++new))==Ø)
            eg.code = MEV_BEFEHL;          /*Untermenü ausgew.*/
           else                   /* Hotkey eines Hauptmenüs? */
            {
             new = MenIKeyIn( key, hmaltkey, hmenanz );
             if ( new!=-1 && MenIsOk( new ) )
              {          /* Hotkey eines Hauptmenüs betätigt */
               eg.code = MEV_HAUPTMEN;      /* Hauptmen angw. */
               newmenr = new;            /* Nummer des Hauptm. */
              }
             else                             /* auch kein Hotkey */
              eg.code = MEV_NO;               /* nichts passiert */
            }
           break;
        }
      }

     else             /* der Linke Mausknopf wurde niedergedrückt */
      {   /* Bereich ermitteln, in dem sich die Maus befand */
       switch ( MenIGetBereich( &bnr ) )  /* Ber. ermitteln */
        {
         case BER_MAUS :          /* Mausbereich außerhalb Menüs */
          eg.code = MEV_MAUS;
          eg.d.bereich = bnr;                  /* Bereichsnummer */
          break;
```

Listing 3: *(Fortsetzung)*

Listing 3: *(Fortsetzung)*

```
        case BER_HMEN :                                /* Hauptmenü */
         if ( MenIsOk( bnr ) )                    /* Menü enabled? */
          {                                                 /* Ja */
           eg.code = MEV_DRAG;                /* in Drag-Modus w. */
           eg.d.befehl = bnr;         /* Nummer des Hauptmenüs */
           if ( *aufptr = (bnr == menr) )  /* gleiches HM? */
            MenIColorSMen( mptr, nr, FALSE );
          }
         else          /* Menü kann nicht angewählt werden */
          eg.code = MEV_ENDE;
         break;

        case BER_SMEN :                                /* Untermenü */
         MenIColorSMen( mptr, nr, FALSE );
         if ( SubMenIsOk( mptr, bnr ) )
          MenIColorSMen( mptr, bnr, TRUE );
         eg.code = MEV_DRAG;                   /* in Drag-Modus w. */
         eg.d.befehl = menr;           /* Nummer des Hauptmenüs */
         *aufptr = FALSE;  /*kein neuer Aufbau erforderlich*/
         break;

        case BER_NO :          /* kein registrierter Bereich */
         eg.code = MEV_ENDE;                     /* Eingabe beenden */
         break;
       }
     }
    /*-- Resultat der obigen Analyse auswerten ------------*/

    if ( eg.code == MEV_MOVE  &&  new != nr )
     {                                  /* ein neues Untermenü */
      MenIColorSMen( mptr, nr, FALSE );
      MenIColorSMen( mptr, nr = new, TRUE );
     }
   }
  while ( eg.code == MEV_MOVE  ||  eg.code == MEV_NO);

  if (*aufptr = !( eg.code==MEV_DRAG && eg.d.befehl==menr ))
   {                                   /* Menüfenster schließen */
    if ( mptr->anzahl )               /* gibt es Untermenüs? */
     MenICloseSubMen( );            /* Ja, Fenster schließen */
    MenIColorHMen( menr, FALSE );  /* Hauptmenü ausblenden */
    menr = newmenr;                  /* neues Hauptmenü setzen */
   }
 }
while ( eg.code == MEV_HAUPTMEN );

if ( eg.code == MEV_BEFEHL )              /* Befehl ausgewählt? */
 {                                                          /* Ja */
  eg.d.befehl = mptr->knr + new;             /* Nummer setzen */
  if ( mptr->toggles & ( 1 << new ) )        /* Toggle-Menü? */
   mptr->tontoff ^= ( 1 << new );            /* Ja, umschalten */
 }
return eg;                   /* Ereignis-Struktur zurückliefern */
}


/***************************************************************
*  Funktion          : M e n I D i s p a t c h                *
**------------------------------------------------------------**
*  Aufgabe           : Steuert die Auswahl eines Untermenüs   *
*                      über Tastatur und/oder Maus.           *
*  Eingabe-Parameter: MODUS = Eingabemodus                    *
*                             ( MEV_POINT oder MEV_DRAG )     *
*                      HMNR  = Nummer des Hauptmenüs          *
*  Return-Wert       : Ereingisstruktur mit Informationen     *
*                      über das eingetretene Ereignis         *
***************************************************************/

MEV MenIDispatch( BYTE modus, BYTE hmnr )
{
 BYTE firsttime;                            /* erster Aufruf? */
 MEV  eg;                      /* zurückgeliefertes Ereignis */
 BYTE spalte,                        /* aktuelle Cursorspalte */
      zeile;                          /* aktuelle Cursorzeile */

 spalte = AKTS;               /* aktuelle Cursorspalte holen */
 zeile = AKTZ;                 /* aktuelle Cursorzeile holen */
 VioHideCursor();            /* Bildschirmcursor verstecken */
 eg.code = modus;                                /* Modus merken */
 eg.d.befehl = hmnr;            /* Nummer des Hauptmenüs merken */
 firsttime = TRUE;           /* erster Aufruf der Funktionen */

 do                                           /* Auswahlschleife */
  {
   if ( eg.code == MEV_DRAG )                        /* Drag-Modus? */
    eg = MenISmDrag( eg.d.befehl, &firsttime );          /* Ja */
   else                                     /* Nein, Point-Modus */
    eg = MenISmPoint( eg.d.befehl, &firsttime );
   /*firsttime = FALSE;    */   /* nicht mehr erster Aufruf */
  }
 while ( eg.code == MEV_POINT  ||  eg.code == MEV_DRAG );

 VioSetCursor( spalte, zeile );         /* Cursor zurücksetzen */
 return eg;                 /* Ereignis-Struktur zurückliefern */
}

/***************************************************************
*  Funktion          : M e n I C h o o s e H M e n            *
**------------------------------------------------------------**
*  Aufgabe           : Wird nach der Auswahl eines Menue-     *
*                      punktes aus dem Hauptmenue aufgerufen  *
*                      und verwaltet die weiteren Eingaben.   *
*  Eingabe-Parameter: keine                                   *
*  Return-Wert       : Ereingisstruktur mit Informationen     *
*                      über das eingetretene Ereignis         *
***************************************************************/

static MEV MenIChooseHMen( void )
{
 MEV   eg;               /* nimmt Ereignis-Code und Daten auf */
 TASTE key;                      /* dient der Tastaturabfrage */
 int   event;                                /* Ereignismaske */
 BYTE  nr,                   /* Nummer des aktuellen Hauptmenüs */
       bereich,                   /* Bereichsnummer für Maus */
       code;                 /* zeigt an, was weiter geschieht */
 int   new;                                 /* neue Menünummer */

 nr = Ø;
 MenIColorHMen( nr, TRUE );      /* Menü als ausgewählt mark. */
 eg.code = MEV_NO;                         /* noch nichts passiert */

 do                                           /* Ereignisschleife */
  {
   /*-- auf Ereignis von Maus oder Tastatur warten ---------*/
   event = KbmEventWait( EV_KEY_AVAIL  | EV_LEFT_PRESS );

   if ( event & EV_KEY_AVAIL )                 /* Taste betätigt? */
    {                                                       /* Ja */
     /*-- Tastencode auswerten ----------------------------*/
     eg.code = MEV_MOVE;             /* von neuem Menü ausgehen */
     switch ( key = KbdGetKey() )                 /* Taste holen */
      {
       case ESC    :                          /* Funktion beenden */
        eg.code = MEV_ENDE;
        break;

       case CR     :                            /* Menü ausgewählt */
        if ( (hmenptr+nr)->anzahl )  /* gibt es Untermenüs? */
         {                                                  /* Ja */
          eg.code = MEV_POINT;            /* in den Point-Modus */
          new = nr;                               /* Nummer merken */
         }
        else                            /* Nein, keine Untermenüs */
         {                                                /* Nein */
          eg.code = MEV_BEFEHL;
          eg.d.befehl = (hmenptr+nr)->knr;
         }
        break;

       case CHOME :                      /* erstes Menü aktivieren */
        new = MenIHmSearch( -1, +1 );
        break;

       case CEND  :                     /* letztes Menü aktivieren */
        new = MenIHmSearch( hmenanz, -1 );
        break;
```

Listing 3: *(Fortsetzung)*

Listing 3: *(Fortsetzung)*

```
      case SPACE :                    /* Menü rechts aktivieren */
      case CRIGHT:
       new = MenIHmSearch( nr, +1);
       break;

      case BS    :                    /* Menü links aktivieren */
      case CLEFT :
       new = MenIHmSearch( nr, -1);
       break;

      default    :                                /* Buchstabe? */
       new = MenIKeyIn( key, hmnorkey,hmenanz );
       if ( new!=-1 && MenIsOk( new ) )
        eg.code = MEV_POINT;            /* in den Point-Modus */
       else                /* testen, ob Hotkey über ALT-Taste */
        {
         new = MenIKeyIn( key, hmaltkey, hmenanz);
         if ( new!=-1  &&  MenIsOk( new ) )
          eg.code = MEV_POINT;                /* in Point-Modus */
         else                       /* unbekannter Buchstabe */
          eg.code = MEV_NO;                  /* nichts passiert */
        }
     }
   }
  else

   /*-- Mausereignis auswerten ----------------------------*/

   switch ( MenIGetBereich( &bereich ) )
    {
     case BER_MAUS :                           /* Mausbereich */
      eg.code = MEV_MAUS;                 /* Funktion beenden */
      eg.d.bereich = bereich;              /* Feldnr. merken */
      break;

     case BER_HMEN :                            /* Hauptmenü */
      if ( MenIsOk( bereich ) )            /* Menü anwählbar? */
       {
        eg.code = MEV_DRAG;                /* in den Drag-Modus */
        new = bereich;                     /* Menünr. merken */
        break;
       }

      case BER_NO :                     /* unbekannter Bereich */
       eg.code = MEV_ENDE;             /* Menüauswahl beenden */
       break;
    }

  /*-- auf das Ereignis reagieren ------------------------*/
  if ( eg.code == MEV_MOVE  &&  new != nr)
   {                                           /* neues Menü */
    MenIColorHMen( nr, FALSE ); /*aktuelles Menü ausblenden*/
    MenIColorHMen( nr = new, TRUE );    /* neues einblenden*/
   }
 }
 while ( eg.code == MEV_NO  ||  eg.code == MEV_MOVE );

 MenIColorHMen( nr, FALSE );                 /* Menü ausblenden */

 if  ( eg.code == MEV_DRAG )      /* in Drag-Modus wechseln? */
  eg = MenIDispatch( MEV_DRAG, new );                   /* Ja */
 else
  if  ( eg.code == MEV_POINT )  /* in Point-Modus wechseln? */
   eg = MenIDispatch( MEV_POINT, new );                 /* Ja */

 MenINewMen( KEIN_MENU );              /* kein Menü mehr aktiv */

 return eg;                /* Ereignis-Struktur zurückliefern */
}

/*************************************************************
* Funktion          : M e n G e t                            *
**----------------------------------------------------------**
* Aufgabe           : Menüverwaltung und -auswahl.           *
* Eingabe-Parameter: keine                                   *
* Return-Wert       : Struktur, die das Ereignis beschreibt  *
* Info              :                                        *
*************************************************************/
```

Listing 3: *(Fortsetzung)*

```
MEV MenGet( void )
{
 MEV   eg;              /* beschreibt das eingetretene Ereignis */
 TASTE key;                      /* dient der Tastaturabfrage */
 int   event;                               /* Ereignismaske */
 BYTE  bereich;                     /* Bereichsnummer für Maus */
 int   i;                                  /* Schleifenzähler */

 MenINewMen( KEIN_MENU );         /* noch kein Menü angewählt */
 eg.code = MEV_ENDE;              /* kein Ereignis verfügbar */

 while ( eg.code == MEV_ENDE )              /* Abfrageschleife */
  {
   /*-- auf Ereignis von Maus oder Tastatur warten ---------*/

   event = KbmEventWait( EV_KEY_AVAIL | EV_LEFT_PRESS );

   if ( event & EV_LEFT_PRESS )     /* Maus-Button gedrückt? */
    {                                /* Ja, Bereich auswerten */

     switch( MenIGetBereich( &bereich ) )
      {
       case BER_HMEN    :                /* Hauptmenü angewählt */
        if ( MenIsOk( bereich ) )             /* Menü enabled? */
         eg = MenIDispatch( MEV_DRAG, bereich );        /* Ja */
        break;

       case BER_MAUS  :                 /* Mausbereich angewählt */
        eg.code = MEV_MAUS;                 /* Funktion beenden */
        eg.d.bereich = bereich;              /* Feldnr. merken */
        break;
      }
    }
   else
    /*-- es wurde eine Taste betätigt ---------------------*/
    {
     if ( (key = KbdGetKey()) == hmkey )        /* Hauptmenü? */
      eg = MenIChooseHMen() ;                           /* Ja */
     else                                             /* Nein */
      {              /* auf Hotkey für eines der Menüs testen */
       if ( (i=MenIKeyIn( key, hmaltkey, hmenanz)) !=-1  &&
            MenIsOk( i ) )
        eg = MenIDispatch( MEV_POINT, i ); /* M. aktivieren*/
       else              /* Taste führt nicht zur Aktivierung */
        {         /* die Taste an den Aufrufer zurückliefern */
         eg.code = MEV_TASTE;              /* Art des Ereignis */
         eg.d.key = key;                 /* die Taste übergeben */
        }
      }
    }
   MenINewMen( KEIN_MENU );        /* kein Menü mehr angewählt */
  }
 return eg;             /* die Ereignis-Struktur zurückliefern */
}


/*************************************************************
*  Funktion          : M e n C o n v N a m e B a c k          *
**----------------------------------------------------------**
*  Aufgabe           : Bringt einen Menünamen zurück in sein  *
*                      ursprüngliches C-Stringformat.         *
*  Eingabe-Parameter: NPTR = Pointer auf den String mit dem   *
*                      Namen                                  *
*  Return-Wert       : keiner                                 *
*************************************************************/

void MenConvNameBack( char * nptr )
{
 BYTE ofs;                       /* Offset des Hotkey im Namen */

 if ( GOT_HOTKEY( nptr ) )      /* Hotkey-Pos bereits ber.? */
  {                                                     /* Ja */
   ofs = (BYTE) *nptr;                       /* Hotkey-Pos holen */
   memmove( nptr, nptr+1, ofs );       /* Platz für MARK_CHAR */
   *(nptr+ofs) = MARK_CHAR;         /* MARK_CHAR wieder setzen */
  }
}
```

Listing 3: *(Fortsetzung)*

```
/************************************************************
*  Funktion          : M e n R e p l a c e N a m e          *
**--------------------------------------------------------**
*  Aufgabe           : Mit Hilfe dieser Funktion kann einem  *
*                      Haupt- oder Untermenü nachträglich ein *
*                      anderer Name zugewiesen werden.        *
*  Eingabe-Parameter: NR   = Nummer des Menüs                 *
*                      NPTR = Pointer auf den Namen           *
*  Return-Wert       : keiner                                 *
*  Info              : - es wird davon ausgegangen, daß das   *
*                        Menü über den gleichen Hotkey wie    *
*                        das vorhergehende Menü verfügt und   *
*                        den gleichen Status im Hinblick auf  *
*                        Enable/Disable und die Toggle-Menüs  *
*                        besitzt                              *
*                      - das neue Menü darf nicht breiter     *
*                        sein, als das Menü das es ersetzt    *
*                      - wird ein Hauptmenü durch den Aufruf  *
*                        dieser Funktion verändert, muß die   *
*                        Funktion MenPrintMen() aufgerufen    *
*                        werden, damit die Veränderung sicht- *
*                        bar wird                             *
************************************************************/
void MenReplaceName( BYTE nr, char *name )
{
 RG HMDES *hptr;          /* Pointer auf den Menü-Beschreiber */
 if ( hptr = MenIFindMen( nr ) )                /* Menü suchen */
  {                                     /* Menü wurde gefunden */
   MenIConvName( name );                 /* Namen konvertieren */
   *( (hptr->mena)+(nr-hptr->knr) ) = name;   /*Namen setzen*/
  }
}

/************************************************************
*  Funktion          : M e n R e p l a c e S u b M e n      *
**--------------------------------------------------------**
*  Aufgabe           : Mit Hilfe dieser Funktion kann ein     *
*                      komplettes Untermenü nachträglich ver- *
*                      ändert werden.                         *
*  Eingabe-Parameter: NR   = Nummer des Menüs                 *
*                      MPTR = Pointer auf den neuen Menü-     *
*                             beschreiber                     *
*  Return-Wert       : keiner                                 *
************************************************************/
void MenReplaceSubMen( BYTE nr, HMDES *mptr )
{
 RG HMDES *hptr;          /* Pointer auf den Menü-Beschreiber */
 if ( hptr = MenIFindMen( nr ) )                /* Menü suchen */
  {                                     /* Menü wurde gefunden */
   if ( hptr->anzahl )   /* hatte des alte Menü Untermenüs? */
    {            /* Ja, allokierte Puffer wieder freigeben */
     free( (void *) hptr->awb );          /* Auswahlbuchstaben */
     free( (void *) hptr->smber );   /* Bereiche der Unterm.*/
    }
   MenIConvName( *(mptr->mena) );   /* Hauptmenünamen konv. */
   memcpy( (void *) hptr, (void *) mptr, sizeof( HMDES ) );
   MenIRegisterSMen( hptr-hmenptr );
  }
}

/************************************************************
*  Funktion          : M e n R e g i s t e r                *
**--------------------------------------------------------**
*  Aufgabe           : Registriert das Hauptmenü.             *
*  Eingabe-Parameter: HMPTR  = Pointer auf den Vektor mit     *
*                               den einzelnen Menübeschr.     *
*                      ANZ    = Anzahl der Menüs im Hauptmenü *
*                      RECHTS = Anzahl der Menüpunkte, die am *
*                               rechten Rand der Menüzeile    *
*                               dargestellt werden            *
*  Return-Wert       : keiner                                 *
************************************************************/
void MenRegister( HMDES * hmptr, BYTE anz, BYTE rechts )
{
 static TASTE altcodes[] =            /* Codes der Alt-Tasten */
  {
   ALT_A, ALT_B, ALT_C, ALT_D, ALT_E, ALT_F, ALT_G, ALT_H,
   ALT_I, ALT_J, ALT_K, ALT_L, ALT_M, ALT_N, ALT_O, ALT_P,
```

Listing 3: *(Fortsetzung)*

```
   ALT_Q, ALT_R, ALT_S, ALT_T, ALT_U, ALT_V, ALT_W, ALT_X,
   ALT_Y, ALT_Z
  };
 RG BYTE i;                                 /* Schleifenzähler */
 BYTE    spalte;    /* Spaltenpos. für jeweiliges Hauptmenü */
 BEREICH *lptr;                          /* Laufzeiger in HMBER */
 HMDES   *mptr;               /* Laufzeiger in das Hauptmenü */
 TASTE   *altptr,                    /* Laufzeiger in HMALTKEY */
         *norptr,                    /* Laufzeiger in HMNORKEY */
         merke;                                       /* Dummy */

 if ( berhm )            /* war bereits ein Menü installiert? */
  MenIFreeMem();  /* Ja, die verschiedenen Puffer freigeben */

 hmenanz = anz;            /* Anzahl der Menüeinträge merken */
 hmenptr = hmptr;                /* Pointer auf Vektor merken */
 hmenrechts = rechts;  /* Anzahl der Einträge rechts merken */
 fptr = (BEREICH *) Ø;         /* noch keine weiteren Felder */
 fanz = Ø;                                             /* dito */

 /*-- 1.) Menünamen umwandeln und dabei die Vektoren mit ---*/
 /*--     den Hotkeys zum Aufruf der Menüs aufbauen       ---*/

 hmaltkey = (TASTE *) malloc( hmenanz * sizeof(TASTE) << 1 );
 hmnorkey = hmaltkey + hmenanz;  /* HMNORKEY im gleichen P. */

 for ( i=Ø, altptr=hmaltkey, norptr=hmnorkey, mptr=hmenptr;
       i < hmenanz;
       ++i, ++mptr )
  {                                /* Hotkey des Menüs holen */
   merke = *(norptr++) = MenIConvName( *(mptr->mena) );
   *(altptr++) = altcodes[ merke - KEY('a') ];
  }

 /*-- 2.) Bereiche der einzelnen Menüs des HM ermitteln, ---*/
 /*--     dabei zunächst die Menüs links, dann die Menüs ---*/
 /*--     rechts durchlaufen                               ---*/

 berhm = (BEREICH *) malloc( hmenanz * SB );
 spalte = MEN_LINKS;                    /* Startspalte laden */
 mptr = hmenptr;      /* mit ersten Menübeschreiber beginnen */

 for ( i = Ø, lptr = berhm;      /* Menüs links durchlaufen */
       i < hmenanz - hmenrechts;
       ++i, ++lptr, ++mptr )
  {
   lptr->x1 = spalte;               /* Start & Endspalte ber. */
   lptr->x2 = ( spalte += strlen( *(mptr->mena)+1 ) ) - 1;
   spalte += MEN_ABSTAND;         /* Spalte für nächstes Menü */
   lptr->y1 = lptr->y2 = MEN_ZEILE;
   lptr->ptr_mask = MEN_STD_CUR;        /* Mauspointer setzen */
  }

 spalte = MEN_R_START;                            /* Endspalte */
 lptr = berhm + hmenanz - 1;           /* Vektoren von hinten */
 mptr = hmenptr + hmenanz - 1;          /* nach vorne durchl.  */
 for ( i = Ø;                     /* Menüs rechts durchlaufen */
       i < hmenrechts;
       ++i, --lptr, --mptr )
  {
   lptr->x2 = spalte;                             /* Endspalte */
   lptr->x1 = ( spalte -= strlen( *(mptr->mena)+1 ) ) + 1;
   spalte -= MEN_ABSTAND;         /* Spalte für nächstes Menü */
   lptr->y1 = lptr->y2 = MEN_ZEILE;
   lptr->ptr_mask = MEN_STD_CUR;        /* Mauspointer setzen */
  }

 /*-- ein Kopie des BERHM-Vektors im Vektor BERALL anlegen -*/

 berall = (BEREICH *) malloc( hmenanz * SB );
 memcpy( berall, berhm, hmenanz * SB );

 /*-- 3.) die einzelnen Menüs im Hauptmenü durchlaufen, ----*/
 /*--     dabei Informationen über die Untermenüs auf-  ----*/
 /*--     reiten und in den HMENPTR-Vektor eintragen    ----*/
 for ( i = Ø; i < hmenanz; )        /* Hauptmenüs durchlaufen */
  MenIRegisterSMen( i++ );               /* Untermenü bearbeiten */
}
```

Listing 3: *(Ende)*

```
/***********************************************************************/
/*                    M E N U T I L . C                              */
/*-------------------------------------------------------------------*/
/*    Aufgabe         : Hält die internen Hilfsfunktionen           */
/*                      für das Modul MEN.C bereit.                 */
/*-------------------------------------------------------------------*/
/*    Autor           : MICHAEL TISCHER                             */
/*    entwickelt am   : 22.02.1989                                  */
/*    letztes Update  : 14.03.1989                                  */
/*-------------------------------------------------------------------*/
/*    Erstellung      : siehe MEN.C                                 */
/***********************************************************************/

#define SAA_MEN_INTERN               /* MENUTIL-Modul kompilieren */

/*-- Include-Dateien einbinden ------------------------------*/

#include "vio.h"          /* Zugriff auf die Video-Funktionen */
#include "kbm.h"           /* Zugriff auf Tastatur & Maus */
#include "men.h"          /* eigene Include-Datei einbinden */
#include <stdlib.h>
#include <ctype.h>
#include <memory.h>
#include <malloc.h>
#include <string.h>

/*-- externe Referenzen auf die globalen Variablen aus -----*/
/*-- dem MEN-Modul                                     -----*/

extern BYTE     hmenanz, /* Anzahl der Einträge im Hauptmenü */
                fanz,          /* Anzahl weiterer Mausfelder */
                aktmen,    /* die Nummer des aktuellen Menüs */
                horat,          /* horizontaler Menürahmen */
                verat;           /* vertikaler Menürahmen */
extern HMDES    *hmenptr;    /* Ptr auf den Hauptmenü-Vektor */
extern MENFKT   menfkt;          /* Aufruf bei Menüwechsel */
extern RAHMEN   rz;                     /* Rahmenzeichen */

/*-- Pointer auf Vektoren mit Bereichsbeschreibern ---------*/

extern BEREICH *berhm,                       /* Hauptmenü */
                *fptr,              /* weitere Mausbereiche */
                *berall,  /* Vektor verknüpft berhm und fptr */
                *aptr;  /*Mausber., Hauptmenü, akt. Untermenü*/

/*-- Pointer auf Vektoren mit Hotkeys für das Hauptmenü ----*/
extern TASTE    *hmaltkey,                       /* ALT-Codes */
                *hmnorkey;                 /* Kleinbuchstaben */

extern HURC     farben; /* Farben für Hauptm., Unterm. etc. */

/***********************************************************************
*  Funktion          : M e n I S c h a t t e n                       *
**-----------------------------------------------------------------**
*  Aufgabe           : Verändert die Attribute der Zeichen in *
*                      einem Bildschirmbereich so, daß der    *
*                      Eindruck eines Schattens entsteht.     *
*  Eingabe-Parameter: X1, Y1 = obere linke Fensterecke        *
*                      X2, Y2 = untere rechte Fensterecke     *
*  Return-Wert       : keiner                                  *
***********************************************************************/

void MenISchatten( BYTE x1, BYTE y1, BYTE x2, BYTE y2 )
{
 VEL     * bptr;           /* Pointer auf allokierten Puffer */
 RG VEL *lptr;           /* Laufzeiger in allokierten Puffer */
 int     anz;             /* Anzahl der Zeichen im Bereich */
 BYTE    attr;                    /* das aktuelle Attribut */

 /*-- Bildschirmbereich in Puffer holen -------------------*/

 anz = ( x2 - x1 + 1 ) * ( y2 - y1 + 1);
 lptr = bptr = (VEL *) malloc( anz * sizeof( VEL ) );
 VioGet( x1, y1, x2, y2, (VEL far *) bptr );

 /*-- den Puffer durchlaufen & Attribute umsetzen ----------*/

 for ( ; anz--; ++lptr )
```

Listing 4: *MENUTIL.C*

```
  {                              /* ein Attribut berarbeiten */
   if ( (attr=lptr->h.attribut) & 128 ) /* heller Hinterg.? */
    attr &= ~128;                  /* Ja, Bit 7 ausblenden */
   else
    attr &= 15;               /* Nein, schwarzer Hintergrund */
   if ( attr & 8 )                  /* heller Vordergrund? */
    attr &= ~8;                  /* Ja, dunkler Vordergrund */
   lptr->h.attribut = attr;           /* neues Attribut setzen */
  }

 /*-- Bildschirmbereich zurück in Bildschirm bringen -------*/

 VioPut( x1, y1, x2, y2, (VEL far *) bptr );
 free( (void *) bptr );             /* Puffer wieder freigeben */
}

/***********************************************************************
*  Funktion          : M e n I N e w M e n                           *
**-----------------------------------------------------------------**
*  Aufgabe           : Wird innerhalb des Moduls immer dann    *
*                      aufgerufen, wenn ein neues Menü ge-    *
*                      wählt wurde.                           *
*  Eingabe-Parameter: NR = Nummer des Menüs                   *
*  Return-Wert       : keiner                                  *
***********************************************************************/

void MenINewMen( BYTE nr )
{
 if ( aktmen != nr )          /* hat sich das Menü verändert? */
  {                                                  /* Ja */
   aktmen = nr;                         /* Menünummer merken */
   if ( menfkt )        /* bei Menüwechsel Funktion aufrufen? */
    ( *menfkt )( nr );                           /* Ja, do it */
  }
}

/***********************************************************************
*  Funktion          : M e n I S m S e a r c h                       *
**-----------------------------------------------------------------**
*  Aufgabe           : Liefert Nummer des nächsten Menüs in   *
*                      einem Untermenü.                        *
*  Eingabe-Parameter: MPTR  = Pointer auf den Menübeschr.     *
*                      START = Nummer des Ausgangsmenüs        *
*                      STEP  = Schritt-Richtung                *
*  Return-Wert       : Nummer des neuen Menüs                  *
*  Info              : - die Untermenünummern werden von 1     *
*                        bis HMDES.ANZAHL gezählt.             *
***********************************************************************/

BYTE MenISmSearch( HMDES *mptr, int start, int step )
{
 if ( mptr->anzahl == 0 )              /* gibt es Untermenüs? */
  return 0;                                           /* Nein */

 do                                              /* Suchschleife */
  {
   start += step;               /* Zähler auf nächstes Menü */
   if ( start > mptr->anzahl )   /* letztes Menü überschr.? */
    start = 1;          /* wieder mit dem ersten Menü beginnen */
   else
    if ( start == 0 )          /* hinter erstes Menü gelangt? */
     start = mptr->anzahl;    /* weiter mit dem letzten Menü */
  }
 while ( !SubMenIsOk( mptr, start) );
 return start;                        /* Menünummer zurückliefern */
}

/***********************************************************************
*  Funktion          : M e n I H m S e a r c h                       *
**-----------------------------------------------------------------**
*  Aufgabe           : Liefert Nummer des nächsten Menüs im   *
*                      Hauptmenü.                              *
*  Eingabe-Parameter: START= Nummer des Ausgangsmenüs         *
*                      STEP = Schritt-Richtung                 *
*  Return-Wert       : Nummer des neuen Menüs                  *
***********************************************************************/
```

Listing 4: *(Fortsetzung)*

```
BYTE MenIHmSearch( int start, int step )
{
 do                                              /* Suchschleife */
  {
   start += step;                   /* Zähler auf nächstes Menü */
   if ( start == hmenanz )    /* letztes Menü überschritten? */
    start = 0;          /* wieder mit dem ersten Menü beginnen */
   else
    if ( start == -1 )          /* hinter erstes Menü gelangt? */
     start = hmenanz-1;  /* mit dem letzten Menü fortfahren */
  }
 while ( !MenIsOk( start ) );
 return start;                      /* Menünummer zurückliefern */
}

/*************************************************************
*  Funktion            : M e n I K e y I n                   *
**-----------------------------------------------------------**
*  Aufgabe             : Prüft, ob sich der übergebene Tasten-  *
*                        code, in dem überg. Vektor befindet.  *
*  Eingabe-Parameter: KEY = zur überprüfende Taste          *
*                        BUF = Ptr. auf den Vektor             *
*                        ANZ = Anzahl der Elemente im Vektor   *
*  Return-Wert         : -1, wenn der Code nicht gefunden wurde,*
*                        sonst die Indexposition im Vektor     *
*************************************************************/

int MenIKeyIn( TASTE key, RG TASTE * buf, BYTE anz )
{
 RG BYTE i;                               /* Schleifenzähler */

 for ( i=0; i < anz; ++i, ++buf )           /* Puffer durchl. */
  if ( *buf != NO_HOTKEY  &&  *buf == key )
   break;                                 /* Taste gefunden! */
 return i==anz ? -1 : i;      /* i==anz --> Taste nicht gef. */
}

/*************************************************************
*  Funktion            : M e n I C o l o r S M e n           *
**-----------------------------------------------------------**
*  Aufgabe             : Färbt eines der Menüs innerhalb eines  *
*                        Untermenüs ein.                       *
*  Eingabe-Parameter: MPTR   = Pointer auf den Menübeschr.   *
*                        NR     = Nummer des Menüs (erstes = 1) *
*                        CHOICE = TRUE, wenn das Menü ausge-   *
*                                 wählt wurde.                 *
*  Return-Wert         : keiner                                *
*  Info                : - diese Funktion verarbeitet keine    *
*                          Trennstriche und keine deaktivierten *
*                          Menüs                               *
*************************************************************/

void MenIColorSMen( RG HMDES *mptr, BYTE nr, BYTE choice )
{
 BYTE coln,             /* Farbe für die normalen Zeichen */
      colh,                      /* Farbe für das Hilight */
      pos;               /* Spalte des Hotkey-Buchstaben */

 if ( nr == 0 )                      /* kein Menü anzeigen? */
  return;                      /* Ja, zurück zum Aufrufer */

 if ( choice )                   /* wurde das Menü angewählt? */
  {                                                  /* Ja */
   MenINewMen( mptr->knr+nr );                 /* neues Menü */
   coln = farben.sm.an;          /* Farbe für normale Zeichen */
   colh = farben.sm.ah;                  /* Farbe für Hotkey */
  }
 else                    /* das Menü wurde nicht ausgewählt */
  {
   coln = farben.sm.sn;          /* Farbe für normale Zeichen */
   colh = farben.sm.sh;                  /* Farbe für Hotkey */
  }

 /*-- Menünamen ausgeben & Auswahlbuchstaben hervorheben ---*/

 MouHideMouse();                             /* Maus verstecken */
 VioColor( VL(1), VO(nr), VR(-1), VO(nr), coln );
```

Listing 4: *(Fortsetzung)*

```
 if ( GOT_HOTKEY( *(mptr->mena+nr) ) )   /* gibt es Hotkey? */
  {               /* Ja, Position ermitteln und dann anzeigen */
   pos = VL(1+MEN_SM_LINKS) + (BYTE) **(mptr->mena+nr);
   VioColor( pos, VO(nr), pos, VO(nr), colh );
  }
 MouShowMouse();                         /* Maus wieder anzeigen */
}

/*************************************************************
*  Funktion            : M e n I C o l o r H M e n           *
**-----------------------------------------------------------**
*  Aufgabe             : Färbt eines der Menüs aus dem Haupt-   *
*                        menü ein.                             *
*  Eingabe-Parameter: NR     = Nummer des Menüs (links = 0)  *
*                        CHOICE = TRUE, wenn das Menü ausge-   *
*                                 wählt wurde.                 *
*  Return-Wert         : keiner                                *
*************************************************************/

void MenIColorHMen( BYTE nr, BYTE choice )
{
 RG BEREICH *bptr;              /* Zeiger in den Bereichsvektor */
 BYTE       coln,             /* Farbe für die normalen Zeichen */
            colh,                      /* Farbe für das Hilight */
            pos;              /* Spalte des Hilight-Buchstaben */

 /*-- Farben für die Einfärbung des Menüs festlegen --------*/
 if ( choice )                    /* wurde das Menü ausgewählt? */
  {                                                  /* Ja */
   MenINewMen( (hmenptr+nr)->knr );                /* neues Menü */
   coln = farben.hm.an;          /* Farbe für normale Zeichen */
   colh = farben.hm.ah;                  /* Farbe für Hilight */
  }
 else                    /* das Menü wurde nicht ausgewählt */
  {
   coln = farben.hm.sn;          /* Farbe für normale Zeichen */
   colh = farben.hm.sh;                  /* Farbe für Hilight */
  }

 /*-- Menünamen ausgeben & Auswahlbuchstaben hervorheben ---*/

 bptr = berhm+nr;               /* Pointer auf BEREICH erstellen */
 MouHideMouse();                             /* Maus verstecken */
 VioColor( bptr->x1-1, bptr->y1, bptr->x2+1, bptr->y2, coln );
 pos = bptr->x1 + (BYTE) **( (hmenptr+nr)->mena );
 VioColor( pos, bptr->y1, pos, bptr->y2, colh );
 MouShowMouse();                         /* Maus wieder anzeigen */
}

/*************************************************************
*  Funktion            : M e n I O p e n S u b M e n         *
**-----------------------------------------------------------**
*  Aufgabe             : Öffnet ein Untermenü                  *
*  Eingabe-Parameter: NR = Nummer des zugehörigen Haupt-     *
*                             menüs. (Hauptmenü links = 0)     *
*  Return-Wert         : keiner                                *
*  Info                ; ein Pointer auf den erstellen Bereichs-*
*                        vektor wird in der globalen Variablen  *
*                        APTR abgelegt.                        *
*************************************************************/

void MenIOpenSubMen( BYTE nr )
{
 HMDES   *hptr;              /* Pointer auf den Menübeschreiber */
 BEREICH *bptr;         /* Pointer auf Bereich des Hauptmenüs */
 BYTE     i,                                /* Schleifenzähler */
          col,                          /* Farbe für Menünamen */
          pos;                   /* Spalte des Hotkey-Buchstaben */
 char    **cptr;                  /* Pointer in den Namen-Vektor */

 /*-- das Menü-Fenster öffnen ------------------------------*/

 hptr = hmenptr + nr;        /* Ptr auf Menübeschreiber setzen */
 MenWinOpen( hptr->rx1, hptr->ry1, hptr->rx2, hptr->ry2,
             horat, verat, RC );

 /*-- die einzelnen Menüs ausgeben ------------------------*/
```

Listing 4: *(Fortsetzung)*

```
MouHideMouse();                         /* Mauscursor ausblenden */
for ( i=1, cptr=hptr->mena+1; i<=hptr->anzahl; ++cptr, ++i )
 {
  if ( *cptr == MEN_TRENNER )                     /* Trennstrich? */
   {                                                     /* Ja */
    VioPrint( VL(1), VO(i), RC, FALSE,
              VioStrep( '-', hptr->rx2-hptr->rx1-1 ) );
    VioPrint( VL(Ø), VO(i), RC, FALSE,
              ( horat == DOPRA ) ? "╟" : "├" );
    VioPrint( VR(Ø), VO(i), RC, FALSE,
              ( horat == DOPRA ) ? "╢" : "┤");
   }
  else                                      /* kein Trennstrich */
   {                                     /* Menünamen ausgeben */
    col = SubMenIsOk( hptr, i ) ? farben.sm.sn
                                : farben.sm.disable;
    VioPrint( VL(1+MEN_SM_LINKS), VO(i), col, FALSE,
              GOT_HOTKEY( *cptr ) ? *cptr+1 : *cptr );
    if ( SubMenIsOk( hptr, i )  &&  GOT_HOTKEY( *cptr ) )
     {  /* es muß ein Hotkey angezeigt werden, Spalte erm. */
      pos = VL(1+MEN_SM_LINKS) + (BYTE) **cptr;
      VioColor( pos, VO(i), pos, VO(i), farben.sm.sh );
     }

    /*-- Toggle-Menü und Toggle auf ON? -------------------*/

    if ( ( hptr->toggles & (1 << i) ) &&
         ( hptr->tontoff & (1 << i) ) )
     VioPrint( VL(1), VO(i), col, FALSE, MEN_TOG_CHAR );
   }
 }

/*-- neuen BEREICH-Vektor erstellen und aktivieren --------*/

aptr = (BEREICH *)
        malloc( ( i = fanz + hmenanz+ hptr->anzahl ) * SB );
memcpy( aptr, fptr, SB * fanz );    /* weitere Mausfelder */
memcpy( aptr+fanz, berhm, SB * hmenanz );      /* Hauptmenü */
memcpy( aptr+fanz+hmenanz, hptr->smber, hptr->anzahl * SB );
MouSetBereich( i, aptr );              /* Bereiche aktivieren */

MouShowMouse();                 /* Mauscursor wieder anzeigen */
}

/***********************************************************
* Funktion          : M e n I C l o s e S u b M e n        *
**-------------------------------------------------------**
* Aufgabe           : Schließt ein zuvor über MenIOpenSubMen *
*                     geöffnetes Untermenü.                *
* Eingabe-Parameter: keine                                 *
* Return-Wert       : keiner                               *
***********************************************************/

void MenICloseSubMen( void )
{
 MouHideMouse();                        /* Mauscursor ausblenden */
 free( (void *) aptr );       /* allokierten Puffer freigeben */
 MouSetBereich( hmenanz+fanz, berall ); /*alte Mausbereiche*/
 VioWinClose( TRUE );               /* Fenster wieder schließen */
 MouShowMouse();                /* Mauscursor wieder anzeigen */
}

/***********************************************************
* Funktion          : M e n I F i n d M e n                *
**-------------------------------------------------------**
* Aufgabe           : Sucht ein Menü auf der Basis seiner  *
*                     Nummer.                              *
* Eingabe-Parameter: MENR = Nummer des gesuchten Menüs     *
* Return-Wert       : Pointer auf den HMDES, zu dem das    *
*                     Menü gehört oder Ø, wenn das Menü    *
*                     nicht gefunden wurde.                *
***********************************************************/

HMDES * MenIFindMen( BYTE menr )
{
 RG HMDES *mptr;              /* Laufzeiger in HMENPTR-Vektor */
 RG BYTE  i;                              /* Schleifenzähler */

 /*-- den HMENPTR-Vektor durchlaufen ----------------------*/

 for ( i = hmenanz, mptr = hmenptr; i--; ++mptr )
  if ( menr >= mptr->knr &&
       menr <= ( mptr->knr + mptr->anzahl ) )
   break;                    /* Menü gefunden, Schleife beenden */

 /*-- nicht gefunden, dann NULL-Pointer zurückliefern ------*/

 return i ? mptr : (HMDES *) Ø;
}


/***********************************************************
* Funktion          : M e n I C o n v N a m e              *
**-------------------------------------------------------**
* Aufgabe           : Konvertiert einen Menünamen, so daß  *
*                     das erste Zeichen im Namen die Offset- *
*                     position des Hotkeys angibt.         *
* Eingabe-Parameter: NPTR = Pointer auf den String mit dem *
*                           Namen                          *
* Return-Wert       : - ASCII-Code des Kleinbuchstabens,   *
*                       über den das Menü aufgerufen werden *
*                       kann, oder die Konstante NO_HOTKEY, *
*                       wenn das Menü über keinen HK verfügt *
***********************************************************/

TASTE MenIConvName( char * nptr )
{
 char * pos;                            /* Zeigt auf MARK_CHAR */

 if ( GOT_HOTKEY( nptr ) )         /* Hotkey-Pos bereits ber.? */
  return ( TASTE ) tolower(*(nptr+1+*nptr));            /* Ja */

 if ( ( pos = strchr( nptr, MARK_CHAR ) ) == (char *) Ø )
  return NO_HOTKEY;                 /* Menü hat keinen Hotkey */

 /*-- das Markierungszeichen wurde gefunden, jetzt wird ----*/
 /*-- dieses Zeichen gelöscht und damit im ersten Byte  ----*/
 /*-- Platz für die Offsetposition gemacht               ----*/

 memmove( nptr+1, nptr, pos-nptr );
 *((BYTE *) nptr) = pos - nptr ;           /* Position merken */
 return (TASTE) tolower( *(pos+1) ); /* Taste zurückliefern */
}


/***********************************************************
* Funktion          : M e n I F r e e M e m                *
**-------------------------------------------------------**
* Aufgabe           : Gibt die Puffer wieder frei, die beim *
*                     Aufruf von MenRegister allokiert     *
*                     werden.                              *
* Eingabe-Parameter: keine                                 *
* Return-Wert       : keine                                *
***********************************************************/

void MenIFreeMem( void )
{
 RG HMDES *lptr;               /* Zeiger in den HMENPTR-Vektor */
 RG BYTE  i;                               /* Schleifenzähler */

 /*-- die Puffer in den HMDESs freigeben -----------------*/

 for ( i=Ø, lptr=hmenptr; i < hmenanz; ++i, ++lptr )
  {
   free( (void *) lptr->awb );          /* Puffer mit Hotkeys */
   free( (void *) lptr->smber );  /* Bereich der Untermenüs */
  }

 free( (void *) hmaltkey ); /* Puffer für HM-Hotkeys freig. */
 free( (void *) hmenptr );     /* Puffer mit Infos freigeben */
 free( (void *) berhm );   /* Bereichs-Vektor wieder freig. */
 free( (void *) berall ); /* gemeinsamen Bereichs-V. freig. */

}
```

Listing 4: *(Fortsetzung)*

```
/************************************************************
*  Funktion         : M e n I G e t B e r e i c h           *
**----------------------------------------------------------**
*  Aufgabe          : Wertet den Bereich aus, über dem sich  *
*                     der Mauscursor befindet.               *
*  Eingabe-Parameter: PTR = Pointer auf eine Byte-Variable,  *
*                           in der die Nummer des Bereichs   *
*                           abgelegt wird.                   *
*  Return-Wert      : eine der Konstanten BER_HMEN, BER_SMEN,*
*                     BER_MAUS oder BER_NO (siehe MEN.H)     *
*  Info             : - die zurückgelieferte Bereichsnummer  *
*                       muß im Hinblick auf den Bereich in-  *
*                       terpretiert werden, über dem sich    *
*                       der Mauscursor befindet.             *
*                       BER_HMEN : Nummer des Hauptmenüs     *
*                       BER_SMEN : Nummer des Untermenüs     *
*                       BER_MAUS : Nummer des Mausbereichs   *
*                       BER_NO   : keine Bedeutung           *
************************************************************/

BYTE MenIGetBereich( BYTE * ptr )
{
 BYTE bereich;          /* nimmt die Nummer des Bereichs auf */

 bereich = MouGetBereich();           /* Bereichsnummer holen */
 if ( bereich == KEIN_BEREICH ) /* kein registrierter Ber.? */
  return BER_NO;                                     /* Ja */

 if ( bereich < fanz )                      /* Mausbereich? */
  {                                                  /* Ja */
   *ptr = bereich;   /* keine Umwandlung der Bereichsnummer */
   return BER_MAUS;
  }

 if ( ( bereich -= fanz ) < hmenanz )          /* Hauptmenü? */
  {                                                  /* Ja */
   *ptr = bereich;          /* Nummer des Hauptmenüs zurückl. */
   return BER_HMEN;
  }

 /*-- es muß sich um ein Untermenü handeln -----------------*/

 *ptr = bereich - hmenanz + 1;          /* Nummer des SMen + 1 */
 return BER_SMEN;
}

/************************************************************
*  Funktion         : M e n I R e g i s t e r S M e n       *
**----------------------------------------------------------**
*  Aufgabe          : Legt die erforderlichen Datenstruk-    *
*                     turen zur Registrierung eines Unter-   *
*                     menüs an.                              *
*  Eingabe-Parameter: NR  = Nummer des Hauptmenüs            *
*  Return-Wert      : keiner                                 *
************************************************************/

void MenIRegisterSMen( BYTE nr )
{
 RG BYTE j, k;                            /* Schleifenzähler */
 BYTE    breite,         /* Breite eines Untermenüsfensters */
         spalte,              /* Spalte des aktuellen Menüs */
         endspalte,          /* Endspalte der Untermenüs */
         maxl;                       /* maximale Menübreite */
```

Listing 4: *(Fortsetzung)*

```
 BEREICH *lptr,              /* Ptr in Bereiche der Hauptmenüs */
         *lptr1;            /* Ptr. in Bereiche der Untermenüs */
 HMDES   *mptr;               /* Laufzeiger in das Hauptmenü */
 TASTE   *norptr;               /* Puffer für Unterm.-Hotkeys */
 char    **cptr;          /* Pointer auf Vektor mit Menünamen */

 mptr = hmenptr + nr;          /* Ptr auf Menübeschreiber setzen */
 lptr = berhm + nr;                    /* Ptr in Bereichsvektor */


 if ( mptr->anzahl )              /* hat dieses Menü Untermenüs? */
  {                            /* Ja, die Untermenüs durchlaufen */
   norptr    =          /* Puffer für Untermenü-Hotkeys allok. */
   mptr->awb = (TASTE *)
                    malloc( mptr->anzahl * sizeof( TASTE ) );
   cptr = mptr->mena+1;        /* Adresse Vektor mit Namen-Ptr */


   /*-- die Untermenüs durchlaufen ----------------------------*/

   for ( j = mptr->anzahl, maxl = 0; j--; ++cptr, ++norptr )
    {
     if ( *cptr != MEN_TRENNER )          /* kein Trennstrich? */
      {                            /* Nein, String konvertieren */
       *norptr = MenIConvName( *cptr );
       k = strlen( *cptr+1 );          /* Stringlänge ermitteln */
       if ( !GOT_HOTKEY( *cptr ) )         /* gibt es einen HK? */
        ++k;                   /* Ja, Stringlänge inkrementieren */
       if ( k > maxl )
        maxl = k;                /* bisher längstes Menü gefunden */
      }
    }


   /*-- Ausrichtung des Untermenüs ermitteln ------------------*/

   breite = MEN_G_BREITE + maxl;          /* Breite des Fensters */
   if ( lptr->x1 - MEN_SM_START +
        MEN_G_BREITE + maxl        <= VioGetCols() )
    spalte = lptr->x1 - MEN_SM_START;             /* Ausr. links */
   else              /* Nein, muß nach rechts ausgerichtet werden */
    spalte = VioGetCols() - ( MEN_G_BREITE + maxl );


   /*-- Bereiche der einzelnen Untermenüpunkte bestimmen ---*/

   lptr1 = mptr->smber = (BEREICH *) malloc(mptr->anzahl*SB);
   endspalte = spalte + MEN_BREITE + maxl - 1;
   for ( j=mptr->anzahl, k=MEN_ZEILE+2, cptr=mptr->mena+1;
         j--;
         ++k, ++lptr1, ++cptr )
    {
     lptr1->y1 = lptr1->y2 = k;                        /* Zeile */
     lptr1->x1 = spalte;                 /* Startspalte setzen */
     lptr1->x2 = endspalte;                /* Endspalte setzen */
     lptr1->ptr_mask = MEN_STD_CUR;               /* Mauscursor */
    }
   mptr->rx1 = spalte;                     /* Koordinaten für  */
   mptr->rx2 = endspalte;                  /* den Rahmen um    */
   mptr->ry1 = MEN_ZEILE + 1;              /* Untermenü setzen */
   mptr->ry2 = k;
  }
}
```

Listing 4: *(Ende)*

Gültigkeitsbereiche und Linken in C:

Die Gültigkeitsbereiche von Bezeichnern

Der richtige Gebrauch von Datenstrukturen ist ein Aspekt beim Programmieren eines guten C-Programms. Zum guten Programmieren müssen Sie die Datenstrukturen sowohl vom syntaktischen als auch vom semantischen Gesichtspunkt verstehen. Sie müssen C-Deklarationen gleich gut lesen und schreiben können, ebenso als würden Sie diese in der menschlichen Sprache formulieren und dann in C übersetzen.

Darüber hinaus müssen Sie die semantische Bedeutung der Bezeichner genau kennen, um sie beim strukturierten Programmieren und beim Verbergen von Daten wirkungsvoll einsetzen zu können. Nachdem Sie den Gebrauch kennen, werden Sie Compiler- und Laufzeitfehler, die von den Datenstrukturen in Ihrem Programm herrühren, besser verstehen. Dies kommt letztendlich auch der Fehlersuche zugute.

Wir erörtern die verschiedenen Konzepte des Gebrauchs von Bezeichnern sowohl innerhalb *einer* Quellcodedatei, als auch über verschiedene hinweg. Sie kennen C nicht richtig, wenn Sie nicht alle Attribute, die ein Bezeichner haben kann, richtig verstehen. Das Erlernen dieser Bezeichner führt zu einer besseren und professionelleren Programmierung.

Deklaration oder Definition

Die meiste Zeit benutzen Programmierer sowohl die Bezeichnung Deklaration als auch die Bezeichnung Definition: Das ist jedoch ein Fehler. Eine Deklaration ist eine Beschreibung der verschiedenen Attribute, die ein Bezeichner haben kann. Das heißt also die Größe, die Art und die Eigenschaft. Eine Deklaration reserviert keinen Speicherplatz für diesen Bezeichner, sie teilt nur etwas über diesen mit. Eine Definition, im Gegensatz hierzu, ist eine Deklaration, die Speicherplatz reserviert, wenn das Programm ausgeführt wird. Dies ist ein wichtiger Unterschied der im Laufe des Artikels noch verdeutlicht wird.

Gültigkeitsbereich

C verwendet einige Regeln, die die Sichtbarkeit eines Bezeichners in Ihrem Programm kontrolliert. Diese Sichtbarkeit wird auch der Gültigkeitsbereich eines Bezeichners genannt und bedeutet die Zugriffsmöglichkeit auf diesen Bezeichner von den verschiedenen Teilen des Programms. Zum Beispiel können Funktionsparameter nur innerhalb der Funktion, in der sie deklariert sind, über ihren Namen angesprochen werden. Deshalb ist hier der Gültigkeitsbereich auf die Funktion beschränkt, das heißt, sie sind nur innerhalb der Funktion ansprechbar.

Arten des Gültigkeitsbereichs	Sichtbarkeit des Bezeichners
Funktion	Labels
Datei	Deklarationen oder Typ-Bezeichner außerhalb einer Funktion oder Parameterliste
Block	Deklarationen oder Typ-Bezeichner innerhalb einer Funktion oder Parameterliste
Funktions-Prototypen	Deklarationen oder Prototypen-Typ-Bezeichner innerhalb einer Parameter-Deklaratorliste

Tabelle 1: *Arten von Gültigkeitsbereichen*

Es gibt vier verschiedene Gültigkeitsbereiche: Blöcke, Funktionen, Funktionsprototypen und Dateien. Der Gültigkeitsbereich beginnt beim Eintritt in einen dieser vier Bereiche und endet beim Verlassen desselben. Zum Beispiel beginnt die Gültigkeit eines Bezeichners mit Blockgültigkeit am Beginn eines Blocks und endet am Blockende.

Tabelle 1 zeigt diese Unterschiede. Die erste Spalte listet die Gültigkeitsbereiche und die zweite zeigt an, für welche Bezeichner Ihres Programms diese in Frage kommen. Wir werden diese vier im Detail besprechen.

Gültigkeitsbereich Funktion

Der einzige Bezeichner, der Funktionsgültigkeit besitzt, ist das Label. Da Labels in einer Funktion eindeutig sein müssen, dürfen sie innerhalb der Funktion höchstens einmal vorhanden sein. Sie können denselben Namen in einer anderen Funktion ohne Namenskonflikt benutzen. Sie werden sehen, daß dies kein Problem darstellt, wenn wir das Linken besprechen.

Gültigkeitsbereich Datei

Bezeicher mit Dateigültigkeit sind diejenigen außerhalb jeder Parameterliste oder Funktion. Die Gültigkeit beginnt am Ende der Deklaration und endet erst am Ende dieser Quellcodedatei. Das bedeutet, daß ein Bezeichner mit Dateigültigkeit nach seinem Gültigkeitsbeginn, von allen Variablen und Funktionen unterhalb, aber nicht oberhalb verwendet werden kann.

Das Beispiel in *Abbildung 1* verdeutlicht dies. Beachten Sie, daß die Variable `file_scope`, obwohl sie außerhalb aller Funktionen erscheint, innerhalb `main` nicht bekannt ist, da sie erst hinterher definiert wird. Mit anderen Worten, ihre Gültigkeit beginnt an der Stelle, an der die Definition steht. Nur nach der Definition von `file_scope` kann von anderen Programmteilen darauf zugegriffen werden.

```
void  f();

main()
{
      f();
      printf("%d\n", file_scope); /* Fehler */
}

int   file_scope = 1;

void  f()
{
      printf("%d\n", file_scope); /* Richtig */
}
```

Abbildung 2: *Beispiel für Gültigkeitsbereich Datei.*

Es stellt sich die interessante Frage, ob es sinnvoll ist `file_scope` nach `main()` und `f()` zu schreiben, unabhängig davon, ob `file_scope` in dieser Datei referenziert wird oder nicht. Ein Teil dieser Antwort wird im Laufe des Artikels gegeben.

Gültigkeitsbereich Block

Der Block-Gültigkeitsbereich ähnelt dem Datei-Gültigkeitsbereich, aber die Grenzen sind etwas enger. Ein Bezeichner mit Blockgültigkeit erscheint innerhalb eines Blocks, das heißt innerhalb geschweifter Klammern ({}), oder ist ein Parameter. Variablen mit Blockgültigkeit werden als lokale Variablen bezeichnet, da sie nur in der Funktion bekannt sind, in der sie erscheinen. Diese Beschreibung von Blockgültigkeit ist zwar richtig, jedoch verwirrend im Zusammenhang mit »externen lokalen Variablen«. Dies wird später bei der Diskussion der Speicherklasse noch deutlich.

Gültigkeitsbereich Funktionsprototypen

Der Funktionsprototypen-Gültigkeitsbereich ist am einfachsten zu verstehen. Es ist nur ein Bezeichner, der in einem Funktionsprototyp deklariert ist. Die Gültigkeit endet, wenn die Funktionsdeklaration endet. Obwohl ein Bezeichner in einem Funktionsprototyp nur optional vorkommt, ist dies hilfreich, da es eine Art Kommentar darstellt, und hierdurch das Programm lesbarer wird.

Beachten Sie aber, daß ein Bezeichner mit Funktionsprototypen-Gültigkeit nur vorliegt, wenn es sich um einen deklarierenden oder vorwärts referenzierenden Prototyp handelt. Ist es ein definierender Prototyp (die eigentliche Funktion selbst), dann sind diese Bezeichner gültige Parameter, die Blockgültigkeit besitzen.

Besonderheiten des Gültigkeitsbereichs

Um den Gültigkeitsbereich zu seinem Vorteil einsetzen zu können, muß man noch einige Dinge wissen. Bezeichner betreten und verlassen den Gültigkeitsbereich. Sehen Sie sich dazu Programm und Ausgaben in *Abbildung 2* an.

```
1   int  id = 5;
2
3   main()
4   {
5     int  id = 4;
6
7     printf("id bei a=%d\n", id);
8     f();
9     {
10        int id = 3;
11
12        printf("id bei b=%d\n", id);
13    }
14    printf("id bei c=%d\n", id);
15  }
16
17  f()
18  {
19    printf("id bei d=%d\n", id);
20  }
```

Ausgabe:

```
id bei a=4    lokale Variable in main - deklariert in Zeile 5
id bei d=5    Variable mit Dateigültigkeit obwohl f() von Zeile
              8 aufgerufen wird - deklariert in Zeile 1
id bei b=3    lokale Variable innerhalb des Blocks in main -
              deklariert in Zeile 10
id bei c=4    lokale Variable innerhalb main - deklariert in
              Zeile 5
```

Abbildung 2: *Ein Beispiel dafür, wie eine Variable in einen Gültigkeitsbereich heraus- und wieder hineintritt.*

In Zeile 1 wird `id` gültig (Datei-Gültigkeitsbereich, um genau zu sein). Dies ist bis zur Zeile 5 der Fall. Hier ist ein anderer Bezeichner mit demselben Namen und derselben Blockgültigkeit. Der vorherige Bezeichner wird jetzt bis zum Ende der Gültigkeit dieses Bezeichners ungültig. Hier ist es das Ende der Funktion (Zeile 15). Ab Zeile 16 ist der Bezeichner `id` mit der Dateigültigkeit wieder sichtbar und wird in Zeile 19 korrekt ausgedruckt, so als wäre kein anderer vorhanden.

Beachten Sie, was sich in Zeile 10 ereignet. Ein neuer Block innerhalb der `main`-Funktion beginnt, kenntlich durch die Klammerung in geschweiften Klammern. Dies ist gültig in C und erlaubt Ihnen Ihre Bezeichner näher an der Stelle zu deklarieren/definieren, an der sie eigentlich benötigt werden. Ebenso ist es in einigen Fällen möglich, den Stack effizienter zu nutzen, oder im Falle von lokalen Variablen oder Registervariablen innerhalb des Blocks effizienteren Code zu erzeugen. So wird in Zeile 10 ein neuer Bezeichner `id` gültig, auf der dritten Gültigkeitsebene, da die beiden Bezeichner `id` von den Zeilen 1 und 5 warten, bis sie wieder gültig werden. `id` von Zeile 10 wird gültig, bis in Zeile 13 sein Gültigkeitsbereich endet. Wiederum referenziert `id` einen neuen Bezeichner.

Linken

Das Linken ist ein Mechanismus, wobei die in Ihrem Programm deklarierten Bezeichner (Objekte oder Funktionen)

unabhängig von der Gleichartigkeit oder Verschiedenheit der Gültigkeit aufgelöst werden. Die Lösung ist das Aufspalten jeder Übersetzungseinheit, das heißt Quellcodedatei, so daß die gegebenen Objekte oder Funktionen, seien sie eindeutig oder nicht, erkannt und mit den Bibliotheksfunktionen gelinkt werden können, um eine ablauffähige Binärdatei zu erzeugen.

Ebenso wie bei dem Gültigkeitsbereich gibt es beim Linken verschiedene Ebenen. Externes, internes oder gar kein Linken. *Tabelle 2* gibt einen Überblick.

Externes Linken

Ein Bezeichner für externes Linken besitzt entweder eine externe Speicherklasse, oder ist ein Bezeichner mit Dateigültigkeit, der nicht explizit `static` deklariert wurde. Jede Referenz zu einem Bezeichner für externes Linken ist eine Referenz zu dem gleichen Objekt oder der gleichen Funktion, egal, von wie vielen oder welchen Quellcodedateien er referenziert wird.

Daher hat der Bezeichner `ext` in *Abbildung 3* das Attribut externes Linken. Wenn die beiden Routinen `main()` und `f()` kompiliert und gelinkt werden, benützen sie denselben Bezeichner `ext`. Wenn Sie also das Programm ausführen, erhalten Sie ein vorhersagbares Ergebnis, wie gezeigt. Beachten Sie, daß sich die Variable vom Punkt a zum Punkt b nicht geändert hat. Nachdem aber `f()` den Wert auf 2 setzt, bleibt er bestehen, bis eine andere Zuweisung ihn verändert, was bis zum Punkt d nicht der Fall ist.

Eine Definition eines nicht statischen Bezeichners mit Datei-Gültigkeitsbereich ist implizit extern. Bedenken Sie aber, daß es innerhalb aller Quellcodedateien nur eine Definition des Bezeichners geben darf. Viele Compiler vernachlässigen diese Regel.

Arten des Linken	**Bezeichnerbeschränkungen**
Extern	Gültigkeitsbereich Extern oder Datei
Intern	Gültigkeitsbereich Datei, aber statisch
Keine	Parameter, keine Objekte, keine Funktionen, keine externen in Blöcken

Tabelle 2: *Linkarten*

```
extern int  ext;
void        f();

main()
{
    ext = 1;
    printf("ext am Punkt a=%d\n", ext);
    f();
    printf("ext am Punkt d=%d\n", ext);
}

Funktion f( ) befindet sich in einer anderen Datei:

extern int  ext;

void f()
{
    printf("ext am Punkt b=%d\n", ext);
    ext = 2;
    printf("ext am Punkt c=%d\n", ext);
}
```

Ausgabe:

```
    ext am Punkt a=1
    ext am Punkt b=1
    ext am Punkt c=2
    ext am Punkt d=2
```

Abbildung 3: *Beispiel für externes Linken.*

Internes Linken

Bezeichner für internes Linken gleichen denjenigen für externes Linken sehr. Die Ähnlichkeit beruht darauf, daß beide Bezeichner mit Dateigültigkeit referenzieren. Der Unterschied besteht darin, daß Bezeichner für internes Linken explizit mit der Speicherklasse `static` deklariert wurden.

```
static int  internal;
void        f();

main()
{
    internal = 1;
    printf("internal am Punkt a=%d\n", internal);
    f();
    printf("internal am Punkt d=%d\n", internal);
}
```

Die Funktion f() befindet sich in einer anderen Datei:

```
static int  internal;

void f()
{
    printf("internal am Punkt b=%d\n", internal);
    internal = 2;
    printf("internal am Punkt c=%d\n", internal);
}
```

Ausgabe:

```
    internal am Punkt a=1     /* internal von main */
    internal am Punkt b=0     /* internal von f    */
    internal am Punkt c=2     /* internal von f    */
    internal am Punkt d=1     /* internal von main
                               * enthält denselben Wert
                               * wie am Punkt a, da kein neuer
                               * Wert zugewiesen wurde */
```

Abbildung 4: *Beispiel für internes Linken.*

unabhängig von der Gleichartigkeit oder Verschiedenheit der Gültigkeit aufgelöst werden. Die Lösung ist das Aufspalten jeder Übersetzungseinheit, das heißt Quellcodedatei, so daß die gegebenen Objekte oder Funktionen, seien sie eindeutig oder nicht, erkannt und mit den Bibliotheksfunktionen gelinkt werden können, um eine ablauffähige Binärdatei zu erzeugen.

Ebenso wie bei dem Gültigkeitsbereich gibt es beim Linken verschiedene Ebenen. Externes, internes oder gar kein Linken. *Tabelle 2* gibt einen Überblick.

Externes Linken

Ein Bezeichner für externes Linken besitzt entweder eine externe Speicherklasse, oder ist ein Bezeichner mit Dateigültigkeit, der nicht explizit `static` deklariert wurde. Jede Referenz zu einem Bezeichner für externes Linken ist eine Referenz zu dem gleichen Objekt oder der gleichen Funktion, egal, von wie vielen oder welchen Quellcodedateien er referenziert wird.

Daher hat der Bezeichner `ext` in *Abbildung 3* das Attribut externes Linken. Wenn die beiden Routinen `main()` und `f()` kompiliert und gelinkt werden, benützen sie denselben Bezeichner `ext`. Wenn Sie also das Programm ausführen, erhalten Sie ein vorhersagbares Ergebnis, wie gezeigt. Beachten Sie, daß sich die Variable vom Punkt a zum Punkt b nicht geändert hat. Nachdem aber `f()` den Wert auf 2 setzt, bleibt er bestehen, bis eine andere Zuweisung ihn verändert, was bis zum Punkt d nicht der Fall ist.

Eine Definition eines nicht statischen Bezeichners mit Datei-Gültigkeitsbereich ist implizit extern. Bedenken Sie aber, daß es innerhalb aller Quellcodedateien nur eine Definition des Bezeichners geben darf. Viele Compiler vernachlässigen diese Regel.

Arten des Linken	Bezeichnerbeschränkungen
Extern	Gültigkeitsbereich Extern oder Datei
Intern	Gültigkeitsbereich Datei, aber statisch
Keine	Parameter, keine Objekte, keine Funktionen, keine externen in Blöcken

Tabelle 2: *Linkarten*

```
extern int  ext;
void        f();

main()
{
     ext = 1;
     printf("ext am Punkt a=%d\n", ext);
     f();
     printf("ext am Punkt d=%d\n", ext);
}

Funktion f( ) befindet sich in einer anderen Datei:

extern int  ext;

void f()
{
     printf("ext am Punkt b=%d\n", ext);
     ext = 2;
     printf("ext am Punkt c=%d\n", ext);
}
```

Ausgabe:

```
     ext am Punkt a=1
     ext am Punkt b=1
     ext am Punkt c=2
     ext am Punkt d=2
```

Abbildung 3: *Beispiel für externes Linken.*

Internes Linken

Bezeichner für internes Linken gleichen denjenigen für externes Linken sehr. Die Ähnlichkeit beruht darauf, daß beide Bezeichner mit Dateigültigkeit referenzieren. Der Unterschied besteht darin, daß Bezeichner für internes Linken explizit mit der Speicherklasse `static` deklariert wurden.

```
static int  internal;
void        f();

main()
{
      internal = 1;
      printf("internal am Punkt a=%d\n", internal);
      f();
      printf("internal am Punkt d=%d\n", internal);
}
```

Die Funktion f() befindet sich in einer anderen Datei:

```
static int  internal;

void f()
{
      printf("internal am Punkt b=%d\n", internal);
      internal = 2;
      printf("internal am Punkt c=%d\n", internal);
}
```

Ausgabe:

```
     internal am Punkt a=1       /* internal von main */
     internal am Punkt b=0       /* internal von f    */
     internal am Punkt c=2       /* internal von f    */
     internal am Punkt d=1       /* internal von main
                                  * enthält denselben Wert
                                  * wie am Punkt a, da kein neuer
                                  * Wert zugewiesen wurde */
```

Abbildung 4: *Beispiel für internes Linken.*

Sie sind vorteilhaft, um Daten zu verbergen. So verwenden zum Beispiel einige der Standardbibliotheksfunktionen wie `fopen`, `fclose` und `malloc` Bezeichner für internes Linken. Der Sinn ist, daß Informationen über diese Bezeichner nicht an den Linker weitergegeben werden, und so nicht in anderen Modulen angesprochen werden können. So sind sie in anderen Quellcodedateien gänzlich unbekannt.

Der Linker behandelt diese, ihm oder auch einem Debugger bekannten Variablen wie einen Kommentar, und sie gelangen nicht in die externe Symboltabelle. Sie können einen internen Bezeichner in einer Quellcodedatei benutzen, ohne sich um Namenskonflikte mit anderen Bezeichnern in anderen Dateien Sorgen machen zu müssen, unabhängig, ob dieser dann extern oder intern ist.

Abbildung 4 zeigt ein Programm mit Bezeichnern für internes Linken. Das Beispiel mag ein wenig schwer zu verstehen sein, da diese Technik nicht oft angewendet wird, aber doch funktioniert. Experimentieren Sie mit dem Beispiel bis Sie das Konzept des internen Linkens verstehen.

Eine kurze Erklärung zu Ihrem Verständnis. Das Programm besteht aus zwei Quellcodedateien, die eine enthält `main()`, die andere `f()`. Wie oben schon erwähnt, sind Bezeichner für internes Linken diejenigen, die Datei-Gültigkeitsbereich besitzen, aber als `static` deklariert sind. Wir wissen auch, daß diese Bezeichner nicht in der externen Symboltabelle des Linkers stehen, wenn dieser externe Verweise auflöst.

Daher hat jede Quellcodedatei in Abbildung 4 seine eigene Kopie des Bezeichners `internal` (es sind also zwei unterschiedliche Ints). Dies wird in der Ausgabe klar, da der Wert an den Punkten a und d gleich ist. Die Ausgabe am Punkt b ist ein wenig trickreicher. Wir erhalten den Wert 0, da alle Variablen der Speicherklasse `static`, unabhängig, ob es sich um internes oder externes Linken handelt, automatisch mit 0 initialisiert werden. Dies ist Standard-C. Beachten Sie zu guter Letzt, daß `internal` in `f()` der Wert 2 zugewiesen wird (wie sie am Punkt c sehen), was aber den Wert, der an Punkt d ausgegeben wird, nicht beeinflußt, da es sich um zwei verschiedene Speicherstellen handelt.

Durch Mischen des Beispiels in Abbildung 4 mit dem in Abbildung 3, könnte sich so etwas ähnliches wie in *Abbildung 5* ergeben. Schauen Sie sich das Programm in Abbildung 5 (BEISP1.C bis BEISP4.C) sorgfältig an, und vergleichen Sie die Ausgaben. Ein paar Kommentare sind angebracht. Die Punkte a, d, g und j zeigen die Variable `messy`, die sich in `main()` befindet. Die Punkte b und c zeigen eine andere Variable `messy`, die nur zu der Funktion x gehört. Die Punkte e, f, h und i zeigen wieder eine andere Variable `messy`, die nur zu den Funktionen y und z gehört. Die Punkte f und h zeigen die gleichen Werte für `messy`, da sie sich auf dieselbe Variable beziehen.

```
/* BEISP1.C */

static int messy;
void   x(), y(), z();

main()
{
     messy = 1;
     printf("a=%d\n", messy);
     x();
     printf("d=%d\n", messy);
     y();
     printf("g=%d\n", messy);
     z();
     printf("j=%d\n", messy);
}
```

```
/* BEISP2.C */

static int messy;

void x()
{
     printf("b=%d\n", messy);
     messy = 2;
     printf("c=%d\n", messy);
}
```

```
/* BEISP3.C */

int messy;

void y()
{
     printf("e=%d\n", messy);
     messy = 3;
     printf("f=%d\n", messy);
}
```

```
/* BEISP4.C */

extern int messy;

void z()
{
     printf("h=%d\n", messy);
     messy = 4;
     printf("i=%d\n", messy);
}
```

Ausgabe:

```
a=1
b=0
c=2
d=1
e=0
f=3
g=1
h=3
i=4
j=1
```

Abbildung 7: *Die Ausgabe ist das Ergebnis der Übersetzung von BEISP1 bis BEISP4 zu einem ausführbaren Programm.*

Kein Linken

Zu guter Letzt gibt es Bezeichner, die nicht gelinkt werden. Parameter und keine Objekte oder keine Funktionen (Typedefs, Strukturelemente und Funktions-Prototyp-Bezeichner) und nicht externe Bereiche in Blöcken.

Dies bedeutet, daß nur Bezeichner mit Dateigültigkeit Kandidaten für das Linken sind. Macht dies Sinn? Im Falle von Strukturelementen und Unionelementen von Typedefs macht es Sinn, da es sie zur Laufzeit nicht mehr gibt, das heißt, es sind keine Objekte. Ebenso existieren Labels, Parameter und Blockbezeichner nur innerhalb Funktionen, weswegen sie typischerweise relativ zum Stack adressiert werden, oder sie sind auf Register bezogen und brauchen nicht adressiert zu werden. Dem Linker braucht also von Ihnen nichts mitgeteilt zu werden. An diesem Punkt versuchen wir eine andere Definition von Linken. Es sollte klar sein, daß keine Notwendigkeit besteht, Verweise aufzulösen, die nicht verbunden werden müssen, da keine andere Quellcodedatei Zugriff hierauf bekommen sollte.

Linkprobleme

Viele der Probleme des Gültigkeitsbereichs können erweitert werden, sobald das Linken von Bezeichnern verstanden wird. Sie haben jetzt eine gute Grundlage, um das Anlegen und die Lebensdauer von Bezeichnern zu verstehen, aber es gibt noch einige Probleme.

Was passiert zum Beispiel, wenn Sie eine Routine haben, die einen Bezeichner anspricht, der sowohl extern als auch intern gelinkt wird? Solch ein Programm wird in *Abbildung 6* gezeigt. Das Beispiel scheint nicht richtig zu sein, aber C läßt es zu. Verschiedene Compilerhersteller behandeln dieses Problem unterschiedlich. Bei einigen funktioniert es. Bei anderen bleibt `ext_int` entweder immer extern oder nie intern oder umgekehrt. Andere behandeln es als Syntaxfehler. Die bevorstehende ANSI-Norm hat dieses Problem erkannt und bezeichnet es als »undefiniertes Verhalten«, Sie sollten also nicht so programmieren.

```
int  ext_int;     /* externes Linken */
void f();

main()
{
    ext_int = 1;
    f();
}

static int ext_int;    /* internes Linken */

void f()
{
    ext_int = 2;
}
```

Abbildung 8: *C gestattet es einer Variablen, sowohl extern als auch intern zu sein. Das Ergebnis ist allerdings vom Compiler abhängig. Der neue ANSI-C-Standard bezeichnet diese Praxis als »undefiniertes Verhalten«.*

Ein weiteres Problem beim Linken entsteht, wenn Bezeichner mit unterschiedlichen Datentypen in den verschiedenen Dateien vorhanden sind. Das kann sich leicht ereignen, ohne daß es Ihnen bewußt wird. Es bereitet Ihnen aber große Schwierigkeiten, wenn Sie nicht verstehen, was Sie getan haben. Ein C-Programmierer (wir hoffen kein besonders geübter in der Sprache C) hat ein Array in seinem Programm und verwechselt die Beziehung zwischen Arrays und Zeigern. Verstehen Sie, weswegen das hier gezeigte Programm compiliert und gelinkt werden kann, aber dennoch nicht richtig arbeitet?

```
int     array[10];
void    func()
main()
{
        <...>
        array[5] = <...>
        <...>
}
```

Eine andere Quellcodedatei enthält folgendes:

```
int     *array;
void    func()
{
        <...>
        array[5] = <...>
        <...>
}
```

Denken Sie ein wenig darüber nach: Die Referenz von `array` in der Quellcodedatei, die `main` enthält, ist richtig. Sehen wir uns aber die Umstände in der anderen Quellcodedatei näher an.

Führt `array[5]` tatsächlich aus, was es ausführen soll? Die Antwort ist nein. Dies beruht darauf, daß `array` als Zeiger umdeklariert wurde. Dies ist ein Problem, da es die Stelle nimmt, auf die `array` zeigt und 5 Integer weiter, und den Wert der Zuweisung an diese Stelle schreibt, das ist aber ein Fehler. Weist der Arrayzeiger eindeutig auf einen Wert? Angenommen es gilt `sizeof(int) == sizeof (int pointer)`, ist dies der Wert `array[0]`.

Hätten wir zum Beispiel das Array in der Quellcodedatei, die `main` enthält, folgendermaßen initialisiert:

```
int array[10] = {
        1, 2, 3, 4, 5, 6,
        7, 8, 9, 10, 11};
```

wäre die Referenz zu `array[5]` in `func *(array + 5)`, was gleichbedeutend mit `*((int *)1 + 5)` ist. Dies ergibt `(int *)1` oder einfach 1, da die Referenzen auf `array` in den beiden Quellcodedateien mit dem Beginn des Arrays aufgelöst werden. Da ein Zeiger eine Variable ist, die die Adresse einer anderen Variablen enthält, ist die Referenz auf `array` in `func` eine Referenz zu dem Objekt, auf das das Array gerade zeigt, in diesem Falle 1.

Bei vielen CPU-Typen fällt dieser Fehler beim ersten Zugriff auf array[5] in func auf, da diese den Zugriff auf eine int-Zahl an einer ungeraden Adresse nicht erlauben. Dies ergibt einen Speicheradressierungsfehler, so daß der Programmierer den Fehler finden kann. Ist jedoch der Wert an der Stelle array[0] ein gerader Wert, ergibt sich kein Fehler. Auf Computern, wie dem IBM PC, auf dem Wortzugriffe auf ungeraden Adressen erlaubt sind, ergibt dies ebenso keinen Fehler. Die Fehlersuche ist in der Regel schwierig, da sich der Programmierer des semantischen Unterschieds der beiden array-Zugriffe in den beiden Dateien nicht bewußt ist.

Ein anderes Problem besteht darin, daß C Groß- und Kleinschreibung unterscheidet, viele Linker hingegen nicht. Das führt zu Situationen, in denen der Linker Kleinschreibung in Großschreibung umwandelt. Das bedeutet aber, daß zwei Bezeichner die SampleIdentifier und sampleidentifier heißen, behandelt werden, als ob es der gleiche Bezeichner wäre - kein besonders günstiger Umstand. Gott sei Dank sind in der Zwischenzeit viele Linker geändert worden, so daß dieses Problem beseitigt ist.

Ein Problem der gleichen Art ist die Länge der signifikanten Stellen der Bezeichner. Viele Compiler erlauben 31 oder 32 Zeichen. Dies ist aber eine syntaktische Einschränkung für den Compiler. Und obwohl Ihr Programm erfolgreich übersetzt wird, kann der Linker die Bezeichner nach sechs Stellen abschneiden.

Ein Bezeichner Bezeichner1 und einer mit dem Namen Bezeichner2 könnten zum Beispiel auf Bezeic gekürzt werden, und stellen so die gleiche Variable dar. Um so schlimmer ist, daß viele Systeme Sie nicht über diesen Umstand unterrichten. Dieses Problem rührt direkt von Ihrem Betriebssystem und Ihren Entwicklungswerkzeugen her. Microsoft C macht das nicht, aber Sie sollten sich dessen bewußt sein, wenn Sie auf ein anderes System portieren.

Ein anderes Problem ergibt sich auf solchen Systemen, die interne Bezeichner in einer Weise an den Linker weitergeben, daß sie mit externen Referenzen aufgelöst werden. Dies bedeutet, daß die Programme in den Abbildungen 4 und 5 nicht in der Weise arbeiten, wie ich es erklärt habe. Es gibt nicht viele Linker, die in dieser Weise arbeiten, und außerdem sind sie in der Regel auf spezialisierten Systemen zu finden. Sollten Sie aber portablen Code schreiben müssen, seien Sie sich dessen bewußt! Wenn Sie viel portieren müssen, empfehle ich Ihnen, ein paar Programme zu schreiben, die das Linkverhalten des Zielsystems untersuchen, bevor Sie die Portierung wirklich vornehmen.

Redeklarationen

Viele C-Compiler mögen keine doppelten Deklarationen oder Definitionen von Bezeichnern. Ich verstehe das nicht, da C dies erlaubt, sofern es sich um dasselbe Objekt, dieselbe Größe und denselben Typ handelt. Diese Diskussion mutet etwas dumm an, da die meisten in der Regel nicht so etwas codieren:

```
int   a;
int   a;
```

Dies kommt aber oft bei Codegenerierungsprogrammen vor. Oder stellen Sie sich vor, Sie inkludieren viele Headerdateien, wobei zwei wiederum dieselbe inkludieren, wie zum Beispiel STDIO.H. Oder nehmen wir den Fall:

```
char   *p1 = &p2;
char   *p2 = &p1;
```

Da die Gültigkeitsregeln von C besagen, daß p2 unbekannt ist, während p1 analysiert wird, können wir also auch nicht die Adresse hiervon erhalten. Die einzige Möglichkeit ist die Vorwärtsreferenz:

```
char   *p2;
```

oder

```
extern char *p2;
```

muß vor p1 stehen.

Einige von Ihnen werden argumentieren, daß C ein Zweipaßcompiler ist, der p2 kennt, wenn der zweite Paß läuft. Erstens gibt es keine Garantie, daß jeder C-Compiler als Zweipaßcompiler implementiert sein muß. Zweitens, obwohl es technisch möglich ist, zu entscheiden, ob p2 existiert (viele Assembler arbeiten in dieser Weise), ist es nicht mit der Definition der Sprache C zu vereinbaren.

Einige dieser Gedanken gehen auf die Tage zurück, als man dachte, es ist am besten, die Dinge so einfach wie möglich zu halten, und den Compiler so schnell wie möglich übersetzen zu lassen. Die Compilerhersteller wollten nicht allzu viele komplizierte Algorithmen implementieren, um den Compiler schnell zu machen. Heutzutage ist die Einstellung etwas anders, aber es besteht der Wunsch, das ursprüngliche C nicht zu verändern.

Lebensdauer

Einen weiteren Punkt habe ich in diesem Artikel vernachlässigt: die Lebensdauer eines Bezeichners. Die Dokumentation von MSC und viele andere beschreiben dies sehr ausführlich, so daß ich hier nur die Aspekte aufgreife, die zu unserer Diskussion passen.

Lassen Sie uns zuerst definieren, was Lebensdauer ist, und wie sie sich zu dem Gültigkeitsbereich und zum Linken verhält. Die Lebensdauer eines Bezeichners gibt an, wie lange er existiert. Es ist wichtig zu bemerken, daß nur Objekte eine Lebensdauer besitzen. Die Dauer eines Objekts ist syntaktisch durch seine Speicherklasse festgelegt, die einen von beiden Werten annehmen kann: statisch oder automatisch. Ein Objekt lebt entweder immer, was bedeutet, daß es statisch ist, oder es lebt sporadisch und dynamisch, während das Programm abläuft, was bedeutet, daß es automatisch ist.

```
void f();
main()
{   extern int i;
    i = 5;
    f();
}

int i;
void f()
{   printf("i=%dØ, i);
}
```

Abb. 7: *Wenn i Datei- oder Blockgültigkeit hat, gibt C i Blockgültigkeit.*

Objekte mit statischer Speicherklasse sind diejenigen zum externen Linken, internen Linken, oder diejenigen die nicht gelinkt werden, aber explizit mit dem Schlüsselwort `static` versehen sind. Objekte mit automatischer Speicherklasse sind diejenigen ohne Linken und ohne explizite statische Speicherklasse.

Sie können sehen, daß die Lebensdauer eines Objekts direkt seine Link-Eigenschaften und in den meisten Fällen auch den Gültigkeitsbereich beeinflußt. Beispielsweise sind alle Bezeichner in Ihrem Programm, die externe oder interne Linkeigenschaften haben, implizit statisch.

Ebenso sind alle mit dem Schlüsselwort `static` versehenen Bezeichner statisch, unabhängig von ihrer Gültigkeit (Datei oder Block). Das heißt, daß alle statischen Objekte vom Programmbeginn bis zum Ende am Leben sind. Alle automatischen Bezeichner leben begrenzt und asynchron, was auch neu oder mehrfach sein kann.

Speicherklassen

Am Ende gibt es noch einige Punkte klarzustellen, nachdem die meisten Eigenarten der Bezeichner erläutert wurden. Zuerst gibt es eine leicht verwirrende syntaktische Eigenheit, die nicht zu dem paßt, was ich gesagt habe. Dies ist die Möglichkeit, Bezeichner mit Datei-Gültigkeitsbereich in einer Art zu referenzieren, daß es scheint, als hätten sie Block-Gültigkeitsbereich, und was schlimmer ist, keine Linkverbindung. Mit anderen

Worten kann es ab und zu vorkommen, daß ein Bezeichner verschiedene Linkverbindungen hat.

Dies ist aber nicht wirklich der Fall. *Abbildung 7* enthält zum Beispiel einen Bezeichner `i` in einer `main()`-Funktion, der keine Dateigültigkeit besitzt, bis die `main()`-Funktion beendet ist. C erlaubt die Abkürzung extern `int i`, um ihn zu referenzieren.

Die Frage ist, ob `i` Block- oder Datei-Gültigkeitsbereich hat? Ohne den Zusatz `extern` ist es ein Bezeichner mit Block-Gültigkeitsbereich und ohne Linkverbindung. Wir wissen jedoch, daß ein Bezeichner mit Datei-Gültigkeitsbereich und externer Linkverbindung referenziert wird. C löst dies, indem es logisch vorgeht. Dem Bezeichner wird Block-Gültigkeitsbereich gegeben. Das erlaubt der Funktion, ohne andere Funktionen die externe Variable anzusprechen.

Zusätzlich zu dieser Überlegung handeln einige Compiler unterschiedlich. MSC gibt allen externen Funktionsdeklarationen Datei-Gültigkeitsbereich und Sichtbarkeit innerhalb des Rests der Quellcodedatei, unabhängig von der Stelle ihrer Deklaration. Dies gilt auch, wenn sie innerhalb eines Blocks deklariert werden, der schon beendet ist. Dies ist keine weitverbreitete Art, aber diejenige, die im ANSI-Standard gefordert wird. Da der Compiler die Information schon hat, gibt es keinen vernünftigen Grund, diese wieder wegzuwerfen.

Die Konzepte des Gültigkeitsbereichs und der Linkverbindung sind nicht allen Programmierern gleich bekannt, wegen der verschiedenen Implementierungen und dem unterschiedlichen Gebrauch. Dies sind aber mächtige Vorteile von C, und falls sie richtig verstanden werden, führen sie zu Programmen, die korrekt integriert werden können und deshalb richtiger arbeiten. Dies ist ein anderer Teil von C, den der Programmierer kennen muß, um professionell zu arbeiten. Er wird um so wichtiger, wenn an größeren Projekten gearbeitet wird, die von mehr als einem Programmierer durchgeführt werden.

Greg Comeau

Eine preiswerte SAA-Oberfläche mit Quellcode:

Die Quick-C-Toolbox

Die Resonanz auf unsere Serie über eine SAA-kompatible Benutzeroberfläche war ausgesprochen gut. Viele Leser warten schon ungeduldig auf die Fertigstellung der kompletten SAA-Toolbox. Dies wird jedoch noch einige Monate dauern. Wir möchten alle Leser, die schon jetzt eine solche SAA-Oberfläche benötigen und nicht mehr warten können auf ein ähnliches Produkt verweisen. Es geht um die Quick-C-Toolbox von Rainer Haselier und Klaus Fahnenstich, die kürzlich im Markt & Technik Verlag erschienen ist. Im folgenden beschreiben die Autoren selbst ihr Buch.

»Mit dieser Toolbox erhalten Sie eine umfangreiche Bibliothek von Funktionen, die Sie bei der Programmentwicklung mit QuickC und Microsoft C verwenden können. Die Tools unterstützen Sie bei der Erstellung einer anwenderfreundlichen Benutzerschnittstelle, die sich an dem von IBM definierten SAA-Standard anlehnt. Durch die Verwendung der Toolbox wird die Dauer für die Erstellung einer Bedieneroberfläche erheblich verkürzt. Zur Verfügung stehen Ihnen:

- Ein professioneller Menümanager, der für Sie Menüzeilen und Pull-Down-Menüs generiert und vom Anwender mit Maus und Tastatur bedient werden kann.
- Ein komfortables Window-Management für die Generierung, Anzeige und Verwaltung von mehreren Fenstern, die sich beliebig überlappen können.
- Eine Sammlung von Dialogfeldern, die Sie über einen simplen Funktionsaufruf in Ihre Programme einbinden können.
- Eine Funktionssammlung zur Unterstützung der Microsoft-Maus.
- Schnelle Routinen zur Bildschirmausgabe, die direkt auf den Bildschirmspeicher zugreifen.

Im Buch finden Sie alle Informationen, die Sie benötigen, um die Microsoft C-/QuickC-Toolbox zu benutzen. Grundlegende Kenntnisse von C und der Bedienung des QuickC-Compilers werden jedoch vorausgesetzt.

In Kapitel 2 werden Sie mit den Grundlagen von SAA, der System Anwendungs-Architektur, bekannt gemacht.

Die Kapitel 3, 4 und 5 führen Sie schrittweise in die wichtigsten Bestandteile der Toolbox ein: den Menümanager, das Window-Management und die Dialogfelder. Beispielprogramme demonstrieren Ihnen, wie Sie die Funktionen in Ihren Programmen einsetzen können.

Kapitel 6 beschreibt die Basisfunktionen der Toolbox, die vor allem die Bildschirmausgabe, die Steuerung der Maus und der Tastatur umfassen. Außerdem finden Sie weitere nützliche Utilities, die auch ohne die in den Kapiteln 3-5 beschriebenen Module in Ihren Programmen eingesetzt werden können.

In Kapitel 7 finden Sie den alphabetisch geordneten Referenzteil der Toolboxfunktionen. Er enthält für jede Funktion neben der Erklärung von Syntax und Parameter auch eine ausführliche Beschreibung ihrer Arbeitsweise. Bei vielen Funktionen sind kleine Beispielprogramme beigefügt, um ihren Einsatz zu verdeutlichen.

Der Anhang soll Ihnen die alltägliche Arbeit mit der Toolbox erleichtern. Sie finden dort eine Tabelle aller Toolbox-Funktionen, eine Liste der Namen aller globalen Variablen der Toolbox, eine Beschreibung der Farbpaletten, eine Tabelle aller Textkonstanten, mit denen das Attributbyte der Bildschirmzeichen bearbeitet werden kann, eine Tabelle aller Textkonstanten, die Sie bei der Bearbeitung von Tastatureingaben unterstützen, und Informationen über die interne Fehlerbehandlung bei kritischen Fehlern.

Zum Buch erhalten Sie vier Disketten, auf denen sich der Source-Code aller Funktionen, fertig kompilierte und gelinkte Einzelbibliotheken und die größeren Beispielprogramme des Buchs befinden.«

Das Buch ist sehr zu empfehlen und hat bei dem Preis von DM 98,- ein exzellentes Preis-/Leistungsverhältnis.

Rainer Haselier, Klaus Fahnenstich: »Quick-C-Toolbox: Eine Einführung in den SAA-Standard mit einer umfangreichen Toolbox zur Realisierung von Benutzeroberflächen«, Haar bei München: Markt & Technik-Verlag, 1989; 290 Seiten; inklusive vier Disketten; ISBN 3-89090-674-5; DM 98,-.



Bild 1: *Ein Beispiel für Menüs, die mit der Quick-C-Toolbox erstellt wurden.*

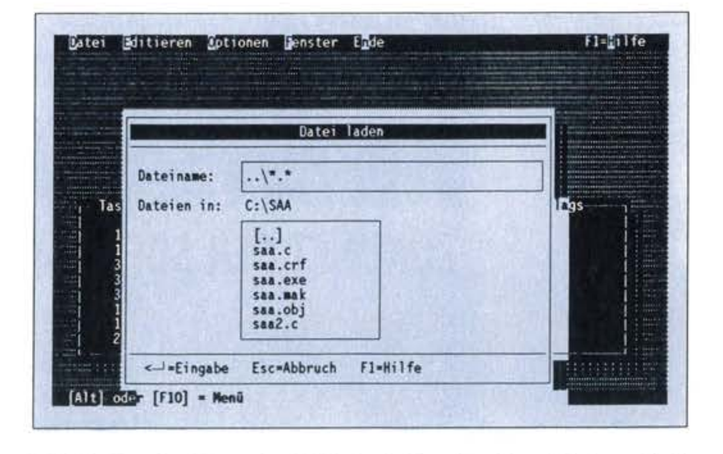


Bild 2: *Es sind Standard-Dialogfelder für Datei laden, Farbeinstellungen und Hilfeinformationen vorhanden.*

Fenster und Fensterklassen unter Windows (Teil 2):

Eigene Dialogfenster-Klassen

Nachdem sich der erste Beitrag (MSJ 3/4 89) dieser kleinen Serie über Fenster und Fensterklassen von Windows mit Unterklassen befaßt hat [1], steht diesmal die Erstellung eigener Fensterklassen im Vordergrund. Mit diesen lassen sich eigenständige Dialogobjekte entwickeln und vielseitig anwenden.

Fensterklassen dienen dazu, die Windows-Programmierung zu vereinfachen, indem oft benötigte Dialogelemente zu Klassen zusammengefaßt werden. Was für einen Aufwand würde es bedeuten, müßte man beispielsweise bei jeder Programmentwicklung das Eingabefeld oder die Rolleiste neu implementieren. Oft sind die vordefinierten Fensterklassen aber nicht ausreichend und auch mit der im ersten Teil vorgestellten Unterklassentechnik lassen sich nicht alle Probleme lösen. Anstatt nun in einem solchen Fall ein benötigtes Dialogobjekt direkt in dem Zeichenbereich einer Applikation abzubilden, ist es viel sinnvoller, das Dialogobjekt als eigenes (Tochter-)Fenster zu definieren und ihm eine eigene Fensterklasse zu geben. Die Fensterklasse charakterisiert dann mit ihrer Implementierung die Eigenschaften des Dialogobjekts, im wesentlichen seine Abbildung und die Art der Kommunikation mit dem Benutzer.

Die Vorteile eigener Fensterklassen

Da Fenster und Applikationen üblicherweise ausschließlich über Nachrichten kommunizieren und diese Nachrichten spezifisch für jede Fensterklasse definiert werden können, besitzen Fensterklassen in Windows eine flexible und komfortable Schnittstelle und die Implementierung kann hinter der Fassade der Nachrichten vollständig verborgen werden. Damit genügen Fensterklassen den Anforderungen der modularen Programmierung.

Eigene Fensterklassen sind in nahezu allen größeren Windows-Applikationen vorhanden. So wurde im *PageMaker* von Aldus das Zeilenlineal als eigene Fensterklasse mit dem Namen »RulerClass« realisiert, beim *Designer* von Micrografx ebenfalls, hier heißt die Fensterklasse des Zeilenlineals »MGX_CAD_RULER«. Am weitesten wurden eigene Fensterklassen in Microsofts *Excel* eingesetzt. Sogar ganz unscheinbare Fenster existieren hier. Beispielsweise gibt es links von und über den Rolleisten eines Tabellenfenster zwei schwarze kleine Flächen. Wird eine dieser Flächen mit der Maus angeklickt und verschoben, kann man die Tabelle in zwei unabhängig betrachtbare Hälften aufteilen. Auch diese zwei Flächen wurden mit einer Fensterklasse namens »XLSPLIT« realisiert. Der Hauptvorteil hier: Der Programmierer muß nicht mehr selbst ermitteln, ob die Maus die Fläche angeklickt hat oder nicht. Dies übernimmt die Fensterverwaltung von Windows und sendet automatisch über das Tochterfenster den Hinweis, daß dieses erfolgt ist.

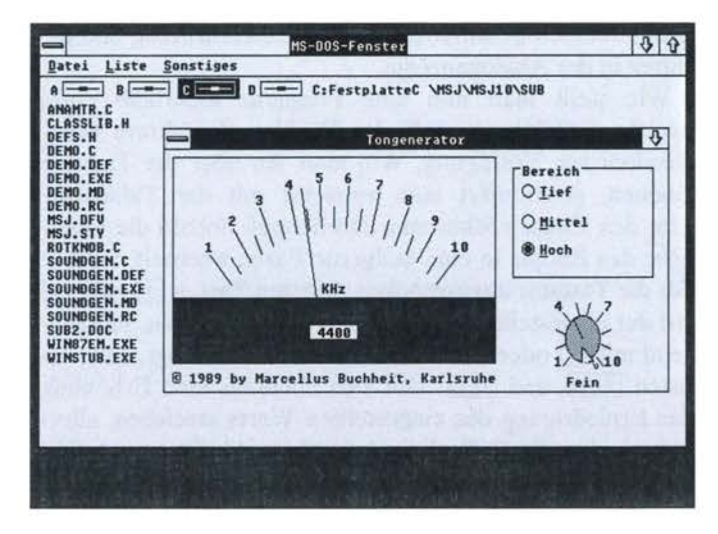


Bild 1: *Die Windows-Applikation SOUNDGEN.*

Dialogobjekte mit eigenen Fensterklassen werden allerdings bis jetzt nur sehr selten in Dialogfeldern (*dialog boxes*) verwendet, obwohl sich gerade über Dialogfelder ein Großteil der Benutzerkommunikation abspielt und die vorhandenen Dialogobjekte in vielen Fällen nicht leistungsfähig genug sind oder zu unergonomisch. Die Verwendung von Dialogfeldern gegenüber der Anordnung der Dialogobjekte mit `CreateWindow` besitzt den erheblichen Vorteil, daß die Definition der Felder vom eigentlichen Programmquellcode getrennt wird und in der Ressourcen-Datei erfolgt. Dadurch können auch weniger erfahrene Programmierer Dialogfelder anlegen oder zumindest überarbeiten (beispielsweise in eine andere Sprache übersetzen). Weiterhin ist die interaktive Erstellung der Felder aufgrund ihrer objektorientierten Definition in der Ressourcen-Datei möglich, auch wenn das dafür vorgesehene Programm DIALOG noch etliche Schwächen besitzt.

Eine Tongenerator-Applikation

Im folgenden wird eine Applikation vorgestellt, die einen Tongenerator simuliert. Man kann eine Frequenz zwischen 10 und 10000 Hertz einstellen und sich den Ton mit der eingestellten Frequenz über den Lautsprecher des PCs anhören. *Bild 1* zeigt das Programm nach dem Aufruf. Es wurde eine Frequenz von 4400 Hertz eingestellt. Wie man sieht, wurden nicht die üblichen Windows-Dialogobjekte verwendet, sondern für die Einstellung der Frequenz innerhalb einer Dekade ein Drehknopf, der mit der Maus oder der Tastatur bedient werden kann und zur Anzeige der Frequenz ein Analogausgabegerät, welches an ein analoges Meßgerät erinnert und über Skala und Zeiger verfügt. Der Wert wird aber zusätzlich auch digital angezeigt, falls es jemand ganz genau haben möchte. Der Frequenzbereich (10 bis 100, 100 bis 1000, 1000 bis 10000 Hertz) wird wie gewöhnlich über runde Optionsfelder (*Radio-Buttons*) eingestellt. Je nachdem, welcher Frequenzbereich gewählt

wurde, wechselt automatisch die Skalenbeschriftung und die Einheit in der Analoganzeige.

Wie stellt man nun eine Frequenz innerhalb eines Bereichs ein? Hierzu stellt der Drehknopf mehrere Möglichkeiten zur Verfügung. Will man ihn über die Tastatur bedienen, positioniert man zunächst mit der Tabulator-Taste den Eingabefokus zum Drehknopf. Sobald die Oberfläche des Knopfs in eine hellgraue Farbe wechselt, wird er über die Tastatur angesprochen. Mit den Tasten [→] oder [↓] wird der eingestellte Wert um eine Einheit erhöht, entsprechend mit [↑] oder [←] um eine Einheit erniedrigt. Mit den Tasten [PgDn] und [PgUp] läßt sich ebenfalls eine Erhöhung oder Erniedrigung des eingestellten Werts erreichen, allerdings um jeweils 10 Einheiten (grobere Abstimmung). Mit [Home] und [End] kann man an den Anfang beziehungsweise das Ende der Skala gelangen. Der erfahrende Windows-Benutzer erkennt die Ähnlichkeit der Bedienung mit der Rolleiste, zumindestens bei der Tastaturbedienung.

Der Drehknopf dreht sich bei jeder Veränderung automatisch mit, indem der Knopfstrich auf den neu eingestellten Wert gesetzt wird. Gleichzeitig erscheint der neue Analogwert am Meßgerät und man kann die gewählte Frequenz für die nächsten 10 Sekunden hören beziehungsweise solange, bis eine andere Frequenz eingestellt wird.

Wesentlich schöner ist es, den Drehknopf mit der Maus zu bedienen. Hierzu sind zwei Varianten vorgesehen: Will man den Drehknopf mit *einem* Maustasten-Druck auf einen gewissen Wert stellen, positioniert man zunächst den Mauspfeil auf den gewünschten Wert in der Knopf-Skala und drückt dann kurz die linke Maustaste. Sofort stellt sich der Drehknopf auf die neue Position ein und die Frequenz ändert sich schlagartig. Die zweite Variante ermöglicht das aktive Drehen des Knopfs. Hierzu positioniert man den Mauspfeil auf eine Position innerhalb der Knopfoberfläche, am besten nahe am Rand und in der Nähe des Strichs, drückt dann die linke Maustaste herunter und hält sie fest. Bewegt man jetzt den Mauspfeil in der Richtung, in der sich der Strich verschieben soll, wird der Knopf sofort in die neue Position gedreht, der Strich rückt nach und die Frequenz wird geändert. Hiermit kann man die Frequenz gleitend erhöhen. Ist der gewünschte Wert eingestellt, läßt man man die Maustaste los und kann die Maus wieder wegziehen.

Verwendung von neuen Fensterklassen

Anstatt die neuen Dialogelemente direkt im Zeichenbereich des Tongenerators abzubilden, wurden zwei neue Fensterklassen mit Namen »AnalogMeter« für das Analog-Meßgerät und »RotationKnob« für den Drehknopf erzeugt. Die Implementierung der beiden Fensterklassen geschieht in den Dateien ROTKNOB.C (*Listing 7*) und ANAMTR.C (*Listing 8*). Doch dazu später. Vorerst wollen wir beide Fensterklassen als »Black Box« betrachten und uns lediglich mit der Anwendung der Fensterklassen beschäftigen.

`RK_SETRANGE`:
: Setzt die neue Knopfposition in *wParam* und die neuen Anfangs- und Endwerte in LOWORD/HIWORD von *lParam*. Alle Knopfpositionen sind vorzeichenlos im Bereich 0 bis 65535.

`RK_GETRANGE`:
: Ermittelt die gesetzten Anfangs- und Endwerte des Knopfs und gibt sie zurück in LOWORD/HIWORD des Rückgabewerts.

`RK_SETPOS`:
: Setzt die neue Knopfposition *wParam* innerhalb des mit `RK_SETRANGE` angegebenen Skalenbereichs.

`RK_GETPOS`:
: Ermittelt die zuletzt gesetzte Knopfposition und gibt sie zurück in LOWORD des Rückgabewerts.

`WM_SETTEXT`:
: Zeichnet eine neue Knopfskala mit den Worten im Fenstertext. Die einzelnen Worte sind mit dem Tab-Zeichen getrennt. Das erste Wort ist die Benennung der Knopfskala (steht unterhalb des Knopfs). Für jedes weitere Wort wird ein Skalenstrich gezeichnet. Das Wort selbst wird an den Skalenstrich passend als Benennung angesetzt, kann aber auch leer sein (unbenannter Skalenstrich). Die Skala selbst umfaßt immer 270°.

Tabelle 1: *Die Steuernachrichten für die Fensterklasse »RotationKnob«*

Listing 1 zeigt den Quellcode für den Tongenerator. Für die vielen neuen Möglichkeiten ist er verhältnismäßig kurz. Das Fenster der Applikation wurde als Dialogfeld spezifiziert. Dadurch wird die gesamte Verwaltung der Nachrichten in das Windows-System verlagert und die Applikation besitzt lediglich eine Funktion zum Bearbeiten der Dialognachrichten. Hier wird die Kommunikation mit den Dialogelementen vollzogen, inklusive der Elemente mit den neuen Fensterklassen. Hierfür wurden teilweise neue Nachrichten definiert.

Innerhalb eines Teams wird man alle nützlichen Fensterklassen in einer Bibliothek sammeln und die Spezifikation der einzelnen Klassen in eine Bibliothek-Kopfdatei verlegen. Obwohl im Beispiel der Einfachheit wegen keine Bibliothek generiert wurde, wurde für die Deklarationen der neuen Fensterklassen bereits eine bibliotheksähnliche Kopfdatei angelegt. Sie ist in *Listing 6* abgedruckt und enthält im wesentlichen die Benennung der neuen Nachrichten für die Steuerung der Fenster. Diese Nachrichten beziehungsweise die Verwendung der `WM_SETTEXT`-Nachricht für die beiden neuen Klassen ist in den *Tabellen 1* und *2* erläutert.

Der Drehknopf besitzt eine ähnliche Programmierschnittstelle wie eine Rolleiste, abgesehen davon, daß man mit `WM_SETTEXT` die Skalenbeschriftung ändern kann.

AM_RANGE:
: Setzt die Anfangs- und Endwerte der Anzeigeskala in LOWORD/HIWORD von *lParam* und die Anzahl der angezeigten Skalenstriche in *wParam*. Der letzte Wert muß im Bereich zwischen 1 und 41 liegen. Der Anfangs- und Endwert muß im Bereich -32768 bis +32767 liegen. Die Skala wird noch nicht neu gezeichnet. Dies geschieht erst nach dem Empfang der WM_SETTEXT-Nachricht.

AM_UPDATE:
: Überträgt einen neuen Anzeigewert in *wParam* an das Anzeigegerät. Der Zeiger wird sofort auf diesen Wert gesetzt. Normalerweise sollte der Wert innerhalb der mit AM_RANGE angegebenen Grenzen liegen. Es wird jedoch eine Überschreitung der Grenzen automatisch erkannt und eine Anzeigebegrenzung vollzogen.

WM_SETTEXT:
: Zeichnet eine neue Anzeigeskala mit den Worten im Fenstertext. Die einzelnen Worte sind mit dem Tab-Zeichen getrennt. Das erste Wort ist die Benennung der Skala (Einheit, steht in der unteren Mitte der Skala). Für jedes weitere Wort wird ein langer Skalenstrich gezeichnet. Das Wort selbst wird an den Skalenstrich passend als Benennung angesetzt, kann aber auch leer sein (unbenannter Skalenstrich). Die Skala selbst umfaßt immer 90°. Die Anzahl der Skalenstriche insgesamt wird mit AM_RANGE festgelegt. Diese Nachricht muß vor WM_SETTEXT an das Fenster geschickt worden sein. Diese Gesamtanzahl minus 1 muß ein ganzzahliges Vielfaches der Skalenwörter im Fenstertext minus 1 sein, sonst wird die Skala nicht bis zum Ende gezeichnet.

Tabelle 2: *Die Steuernachrichten für die Fensterklasse »AnalogMeter«.*

Aber die Rückgabe einer neuen Position des Knopfs erfolgt mit Hilfe von WM_VSCROLL-Nachrichten exakt so wie bei der Rolleiste. Auch bei der Einstellung des aktuellen Anzeigewerts oder des Skalenbereichs hätte ich gerne Nachrichten verwendet, die für die Rolleiste vorgesehen sind. Dadurch wäre ein problemloser Wechsel zwischen einer Rolleiste und dem Drehknopf allein durch Änderung der Dialogfeldspezifikation in der Ressourcen-Datei möglich gewesen, also sehr einfach und flexibel. Leider erfolgt die Einstellung einer Rolleiste nicht über Nachrichten sondern über die System-Funktionen SetScrollPos und SetScrollRange, die irgendwie direkt auf die Fensterklasse der Rolleisten einwirken. Man sieht, welche Folgen eine nicht sauber durchgezogene Systemspezifikation haben kann. Beim OS/2-Presentation-Manager ist die Welt wieder in Ordnung: Hier wird eine Rolleiste mit Hilfe der Nachricht SBM_SETSCROLLBAR eingestellt, die Einführung der Nachrichten RB_SETRANGE, RK_GETPOS etc. wäre nicht erforderlich gewesen.

```
/*************************************************************

 ----  S O U N D G E N  ---- MS-Windows Application ---------

=============================================================

This MS-Windows application is a simulator of a sound
generator. The sound frequency is selected by a rotation knob
and is displayed by an analog meter. The sound can be heared
via the system's loudspeaker (in LOFI-quality).

This program is a small programm for demonstration and study
the Windows programming technique of generating own window
classes.

-------------------------------------------------------------


Copyright 1989 by
          Marcellus Buchheit, Buchheit software research
          Zaehringerstrasse 47, D-7500 Karlsruhe 1
          Phone (0) 721/37 67 76     (West Germany)

Release 1.00 of 89-Mar-18 --- All rights reserved.

*************************************************************/


/* define/undefine non-debugging option name */
#undef   NDEBUG

#define  NOMINMAX
#include <WINDOWS.H>
#include "DEFS.H"
#include "CLASSLIB.H"

/* ANSI-C headers */
#include <STDLIB.H>

/* window function parameter macros */
#define P2LO LOWORD(ulP2)
#define P2HI HIWORD(ulP2)
#define LADDR(r) (LONG)(LPSTR)(r)

/* instance of main dialog procedure */
FARPROC rfdMain;

/* current frequency range (1,10,100) */
int uFreqRange=1;

/* current relative frequency value, always in range 10..100 */
int uFreqValue=10;

/* Flag "sound generator is active" */
BOOL bSoundActive=FALSE;


/*************************************************************
 S e t F r e q R a n g e
=============================================================

This function sets a new range of the panel frequency measure
instrument.

Parameters:
 hw  is the menu handler of the panel window.

Used variables:
 uFreqRange, uFreqValue;

Return:
 none

*************************************************************/

VOID SetFreqRange(HWND hw)

 {HWND hwMeter;
  char *rzScale;
```

Listing 1: *Das Programm SOUNDGEN.C*

```
  hwMeter=GetDlgItem(hw,IDMETER);
  /* set new measure instrument range:
     10*uFreqRange..100*uFreqRange */
  SendMessage
   (hwMeter,AM_RANGE,19,MAKELONG(10*uFreqRange,100*uFreqRange)
   );
  switch (uFreqRange)
   {case 1:
      rzScale="Hz\t10\t20\t30\t40\t50\t60\t70\t80\t90\t100";
      break;
    case 10:
      rzScale=
       "Hz\t100\t200\t300\t400\t500\t600\t700\t800\t900\t1000";
      break;
    case 100:
      rzScale="KHz\t1\t2\t3\t4\t5\t6\t7\t8\t9\t10";
      break;
   } /* switch */
  SetWindowText(hwMeter,rzScale);
  /* set new output value: uFreqValue*uFreqRange */
  SendMessage(hwMeter,AM_UPDATE,uFreqValue*uFreqRange,0L);
 } /* SetFreqRange() */

/****************************************************************
 S e t F r e q V a l u e
=================================================================

This function sends a new value of the frequency to the fine
selection item and to the panel frequency measure instrument.

Parameters:
 hw  is the menu handler of the panel window.

Used variables:
 uFreqValue

Return:
none

****************************************************************/

VOID SetFreqValue(HWND hw)

 {/* set value into scrolling item */
  SendMessage(GetDlgItem(hw,IDFINE),RK_SETPOS,uFreqValue,0L);
  /* set value into analog meter */
  SendMessage
   (GetDlgItem(hw,IDMETER),AM_UPDATE,uFreqValue*uFreqRange,0L);
  /* change sound frequency *7
  if (!bSoundActive)
   {/* open sound generator for application */
    if (OpenSound()<0) return; /* sound module not available */
    bSoundActive=TRUE; /* sound is active */
   } /* if */
  StopSound(); /* stop old sound */
  /* start sound of current frequency for about 10 seconds */
  SetVoiceAccent(1,120,128,S_LEGATO,0);
  SetVoiceSound(1,MAKELONG(0,uFreqValue*uFreqRange),4000);
  StartSound(); /* start new sound */
 } /* SetFreqValue() */

/****************************************************************

 ----------------- main dialog box function ----------------

****************************************************************/

/****************************************************************
 f d M a i n
=================================================================

/// Function for Modal Dialog Box ///

This function processes any messages received by the DBX_LINE
dialog box.

****************************************************************/
```

Listing 1: *(Fortsetzung)*

Abgesehen von den neuen Nachrichten ist die Implementierung der Dialogfeldfunktion ziemlich standardmäßig realisiert. Die Definition des Dialogfelds erfolgt in der Ressourcen-Datei, abgedruckt im *Listing 3.* Man erkennt die Namen der neuen Klassen. Die beiden Dialogelemente werden im übrigen genauso spezifiziert wie Optionsboxen, Textfelder etc. Die angegebene Größe beschreibt die Eckpunkte des Element-Rechtecks und der angegebene Text den Fenstertext. Bei der Analoganzeige wird der Text allerdings von der Dialogfeldfunktion gesetzt und die Initialisierung ist leer. Die Skala des Drehknopfs wird dagegen nie verändert und bereits in der Ressourcen-Datei über den Fenstertext definiert.

In den weiteren Listings von SOUNDGEN steht nichts Besonderes. *Listing 2* zeigt die MAKE-Datei, mit der der Tongenerator übersetzt wird. *Listing 4* enthält die gemeinsamen Definitionen der Ressource-Datei und des C-Quellcodes. *Listing 5* beschreibt die nötigen Definitionen für den Linker. Es kann sowohl der MS-Windows-Linker (Version 5.01.17) als auch der OS/2-Linker (Version 5.01.21) verwendet werden. Zum Übersetzen der C-Listings wurde der Microsoft-C-Compiler, Version 5.10, verwendet.

Da die Fensterklassen mit Fließkommaarithmetik arbeiten, muß die Option `-FP` des Compilers gewählt werden. Es wurde die Emulationsvariante verwendet, die 8087-Befehle generiert und falls kein Arithmetikprozessor vorhanden ist, diese Befehle softwaremäßig emuliert. Hierzu wird unter Windows die dynamische Link-Bibliothek WIN87EM.EXE benötigt. Wenn Sie jemandem SOUNDGEN weitergeben wollen, dürfen Sie nicht vergessen, WIN87EM.EXE mitzukopieren.

Wenn ein Bibliothekseintrag eine Fensterfunktion enthält und dieser Eintrag wird hinzugebunden, muß unbedingt der Namen der Fensterfunktion in der EXPORT-Liste der Linker-Definitionsdatei erscheinen. Damit man leichter daran denkt, habe ich diese Namen auch in `CLASSLIB.H` angegeben. Das manuelle Eintragen der vom Windows-System aufzurufenden Funktionen in die Definitionsdatei ist normalerweise schon lästig genug, aber mit Bibliotheken wird es wirklich zu einem Problem. Es stört empfindlich die modulare Programmentwicklung und verursacht oft unerwartete Abstürze mit langwierigen Debugging-Sitzungen. Hier sollte sich Microsoft unbedingt eine Lösung einfallen lassen. Denkbar wäre es, etwa ein zusätzliches C-Pragma »pragma system_export«, das den Namen in der .OBJ-Datei als besonders exportiert kennzeichnet.

Das Applikationsfenster als Dialogfeld

Auf den ersten Blick sieht die Verwendung eines Dialogfelds als Applikationsfenster sehr praktisch aus. Es ergeben sich aber eine Reihe von Nachteilen. So werden nicht alle Nachrichten, die an ein Dialogfeld geschickt werden, auch an die Dialogfeld-Funktion übergeben. Weiterhin ist es nicht möglich, einer Nachricht einen Rückgabewert als

Ergebnis zuzuweisen, da die Dialogfeld-Funktion nicht das Ergebnis einer Nachricht zurückgibt, sondern einen booleschen Wert, der angibt, ob die Nachricht in der Funktion bearbeitet wurde oder nicht.

Weiterhin läßt sich nicht mit wenig Aufwand einer Dialogfeld-Applikation ein Sinnbild hinzufügen. Die in *[3]* angegebene Lösung funktioniert leider nicht korrekt. Wie wir im ersten Teil dieser Serie erfahren haben, wird das Sinnbild der Fensterklasse und nicht dem einzelnen Fenster zugewiesen. Alle Fenster einer Klasse teilen sich aber die Klassenspezifikationen, diese werden nicht beim Anlegen eines neuen Fensters dupliziert. Fügt man wie in *[3]* dem Dialogfeld ein Sinnbild hinzu, wird in Wirklichkeit der Klasse »#32770« das Sinnbild hinzugefügt und alle Applikationen, die ihr Fenster über eine Dialogbox darstellen, erhalten dieses eine Sinnbild. Sie können dies ausprobieren: Wird SOUNDGEN allein aufgerufen und zu einem Sinnbild gemacht, erscheint dieses als leere weiße Fläche, da ein Dialogfeld kein Sinnbild besitzt. Rufen Sie aber anschließend das Programm PALETTE *[3]* auf, erscheint beim erneuten Verkleinern von SOUNDGEN das Sinnbild von PALETTE, was natürlich nicht sinnvoll ist und vielleicht sogar zu Abstürzen führen kann.

Die Verwendung von Dialogfeldern als Applikationsfenster ist daher auf Demonstrationsbeispiele zu beschränken, die schnell programmiert sein müssen und deren Quellcode sich auf das Wesentliche beschränken soll. Für professionelle Applikationen dagegen, deren Erscheinungsbild einem Dialogfeld sehr ähnelt, ist es unumgänglich, wie gewöhnlich ein Applikationsfenster mit einer applikationseigenen Klasse anzulegen, aber dieser Fensterklasse ein eigenes Dialogfeld fest zuzordnen, indem bei der Definition dieses Felds in der Ressourcen-Datei das CLASS-Statement verwendet wird. Diese Technik ist in *[4]* ausführlich erläutert und wird unter anderem im Programm PIFEDIT verwendet. Und PIFEDIT verfügt bekanntlich über ein korrekt funktionierendes Sinnbild.

Ansteuern des SOUND-Moduls von Windows

Der Tongenerator war meine erste Applikation, welche das SOUND-Modul von Windows benutzte. Dieses Modul ist *der* Teil von Windows, der in der Development-Kit-Dokumentation *[1]* mit Abstand am schlechtesten beschrieben wird. Auch die Autoren der mittlerweile etwas zahlreicher zu erhaltenden Windows-Literatur machen einen weiten Bogen um die Klangerzeugung und gehen mit keinem Wort auf sie ein, obwohl das Modul einige interessante Eigenschaften besitzt. Grund genug, in diesem Artikel kurz darauf einzugehen. Wie so häufig bei exotischen Windows-Eigenschaften driften die Beschreibung im Devolopment-Kit und die Implementierungswirklichkeit teilweise weit auseinander. Man ist auf das teilweise Disassemblieren des SOUND-Moduls angewiesen, will man zu einigermaßen brauchbaren Ergebnissen kommen.

```
BOOL FAR PASCAL fdMain(HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {static HWND hwF;

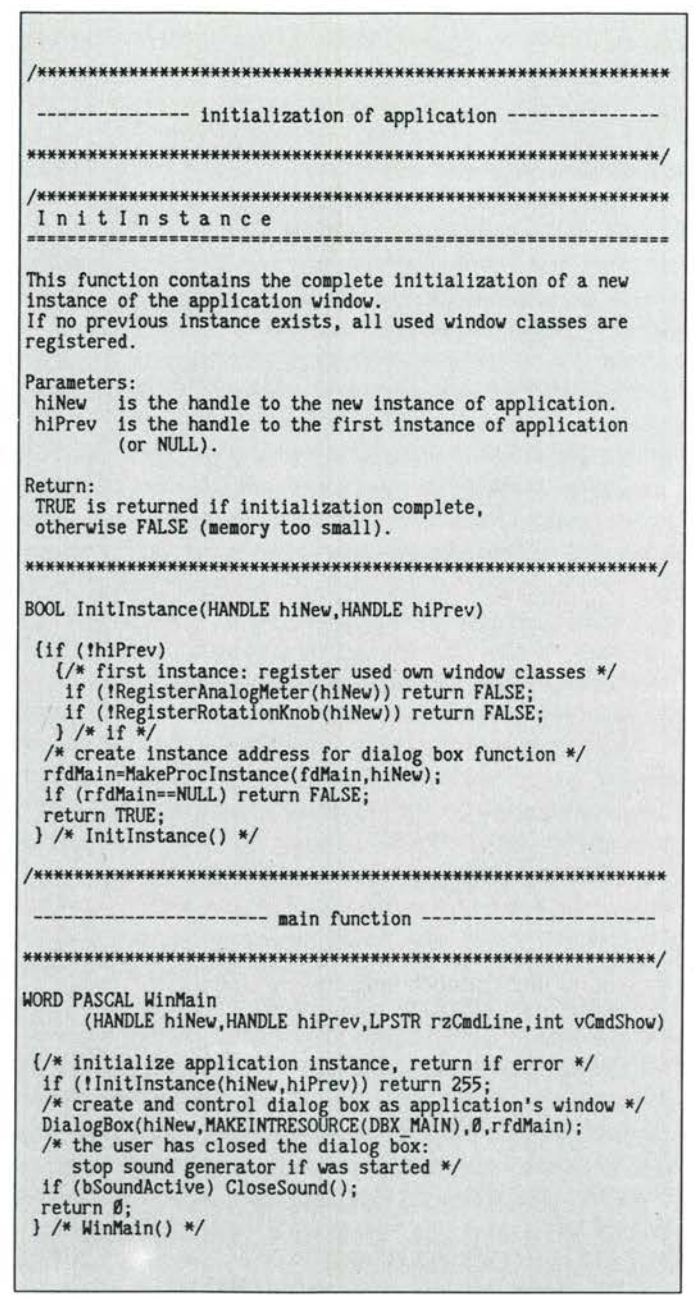
  switch (iMsg)
   {case WM_INITDIALOG:
      /* set last specified range */
      CheckRadioButton(hw,IDRANGE10,IDRANGE1000,IDRANGE10);
      /* set fine selector range 10..100 */
      hwF=GetDlgItem(hw,IDFINE);
      SendMessage(hwF,RK_SETRANGE,10,MAKELONG(10,100));
      /* set range and value of measure instrument */
      SetFreqRange(hw);
      return TRUE;

    case WM_COMMAND:
      switch (uP1)
       {case IDRANGE10:
          uFreqRange=1;
          break;
        case IDRANGE100:
          uFreqRange=10;
          break;
        case IDRANGE1000:
          uFreqRange=100;
          break;
        default:
          return FALSE;
       } /* switch */
      /* set new range, update value */
      CheckRadioButton(hw,IDRANGE10,IDRANGE1000,uP1);
      SetFreqRange(hw); SetFreqValue(hw);
      return TRUE;
      break;

    case WM_HSCROLL:
    case WM_VSCROLL:
     {int uOldFreq=uFreqValue;
      switch (uP1)
       {case SB_BOTTOM:
          uFreqValue=100;
          break;
        case SB_TOP:
          uFreqValue=10;
          break;
        case SB_LINEDOWN:
          uFreqValue+=(uFreqValue<100);
          break;
        case SB_LINEUP:
          uFreqValue-=(uFreqValue>10);
          break;
        case SB_PAGEDOWN:
          uFreqValue+=min(10,100-uFreqValue);
          break;
        case SB_PAGEUP:
          uFreqValue-=min(10,uFreqValue-10);
          break;
        case SB_THUMBTRACK:
        case SB_THUMBPOSITION:
          uFreqValue=P2LO;
          break;
        default:
          return FALSE;
       } /* switch */
      if (uFreqValue!=uOldFreq) SetFreqValue(hw);
      return TRUE;
     } /* block */

    case WM_ENDSESSION:
      /* end of session: stop sound output */
      if (bSoundActive) CloseSound();
      return TRUE;
    case WM_CLOSE:
      /* close dialog box, terminate application */
      EndDialog(hw,FALSE); return TRUE;
   } /* switch */
  return FALSE;
 } /* fdMain() */
```

Listing 1: *(Fortsetzung)*



```
/*************************************************************

 -------------- initialization of application --------------

*************************************************************/

/*************************************************************
 I n i t I n s t a n c e
=============================================================

This function contains the complete initialization of a new
instance of the application window.
If no previous instance exists, all used window classes are
registered.

Parameters:
 hiNew   is the handle to the new instance of application.
 hiPrev  is the handle to the first instance of application
         (or NULL).

Return:
 TRUE is returned if initialization complete,
 otherwise FALSE (memory too small).

*************************************************************/

BOOL InitInstance(HANDLE hiNew,HANDLE hiPrev)

 {if (!hiPrev)
   {/* first instance: register used own window classes */
    if (!RegisterAnalogMeter(hiNew)) return FALSE;
    if (!RegisterRotationKnob(hiNew)) return FALSE;
   } /* if */
  /* create instance address for dialog box function */
  rfdMain=MakeProcInstance(fdMain,hiNew);
  if (rfdMain==NULL) return FALSE;
  return TRUE;
 } /* InitInstance() */

/*************************************************************

 ---------------------- main function ----------------------

*************************************************************/

WORD PASCAL WinMain
      (HANDLE hiNew,HANDLE hiPrev,LPSTR rzCmdLine,int vCmdShow)

 {/* initialize application instance, return if error */
  if (!InitInstance(hiNew,hiPrev)) return 255;
  /* create and control dialog box as application's window */
  DialogBox(hiNew,MAKEINTRESOURCE(DBX_MAIN),Ø,rfdMain);
  /* the user has closed the dialog box:
     stop sound generator if was started */
  if (bSoundActive) CloseSound();
  return Ø;
 } /* WinMain() */
```

Listing 1: *(Ende)*

Das SOUND-Modul wird mit der Funktion `OpenSound` einer Applikation zugewiesen. Da nur ein Lautsprecher pro PC vorhanden ist, wird die Zuweisung abgelehnt, wenn bereits einer anderen Applikation das SOUND-Modul zugewiesen wurde. Entsprechend gibt eine Applikation durch Aufruf der Funktion `CloseSound` das Modul wieder frei. Für die Klangausgabe stehen pro Stimme je ein Puffer zur Verfügung, in den mit `SetVoiceNote` Noten aus 7 Oktaven oder mit `SetVoiceSound` Tonhöhen-Werte samt Zeitdauer und Klang eingegeben werden. Die Anzahl dieser Puffer wird von `OpenSound` zurückgegeben. Bei der primitiven PC-Tonausgabe gibt es natürlich nur eine Stimme und somit auch nur einen Puffer. Er wird bei allen Ausgabefunktionen mit dem Wert 1 angesprochen. Man kann auch den auszugebenden Tönen mit der Funktion `SetVoiceAccent` Akzente geben, beispielsweise ob Legato oder Stakkato gespielt werden soll, das Tempo wählen oder die Lautstärke einstellen.

Da der Tongenerator mit Tonhöhen und nicht mit Noten arbeitet, wurde im Beispiel die Funktion `SetVoiceSound` verwendet, die in den Puffer einen Ton mit einer bestimmten Frequenz und einer Ausgabezeit schreibt. Leider ist die Funktion in der Dokumentation völlig falsch beschrieben oder die Implementierung wurde falsch vorgenommen. Jedenfalls haben Experimente folgendes ergeben: Der Parameter *nFrequency* (er ist vom Typ `long`, nicht `int`) enthält die gewünschte Frequenz in Hertz und der Parameter *nDuration* enthält die gewünschte Länge in den Zeitgeber-Intervallen des Tongenerators. Ein solches Intervall beträgt ungefähr 2,5 ms. Mit dem angegebenen Wert von 4000 in der Funktion `SetFreqValue` des Tongenerators kommen wir somit auf etwa 10 Sekunden Länge.

Die eigentliche Klangausgabe erfolgt im Hintergrund. Sie beginnt mit dem Aufruf der Funktion `StartSound` und bricht sofort ab, wenn man `StopSound` aufruft.

Keinesfalls darf vergessen werden, vor dem Beenden einer Applikation oder des Windows-Systems die Funktion `CloseSound` aufzurufen. Wenn Sie dies nicht befolgen, geschehen merkwürdige Dinge. Denn die Klangerzeugung verändert die Zeitkonstante des internen PC-System-Ticks, der normalerweise auf 1/18 Sekunde eingestellt ist, auf einen deutlich kleineren Wert (etwa 1/400 Sekunde). Sie können dies testen, indem Sie in SOUNDGEN.C die Zeile nach »case WM_ENDSESSION« in der Funktion `fdMain` entfernen und Windows über das MS-DOS-Fenster verlassen, *bevor* die Tonausgabe beendet ist. Der Ton wird auf DOS-Ebene endlos weiter zu hören sein. Die Systemuhr ist plötzlich viel schneller. Wenn Sie jetzt Windows erneut aufrufen (die Maus hat sich wahrscheinlich schon verabschiedet) und die Windows-Uhr starten, erleben Sie, was passiert, wenn Sie sich mit Ihrem Rechner einem schwarzen Loch im Weltall nähern: Die Uhr läuft mit einer ziemlich hohen Geschwindigkeit, sonst allerdings vollkommen korrekt. Wenn Sie sich an dem Phänomen sattgesehen haben, sollten Sie Ihr System neu booten und sich die Ursache für die Zukunft merken.

Erwarten Sie von dem Klangmodul keine Meisterleistungen. Schließlich erfolgt die Ausgabe über einen Lautsprecher der alleruntersten Preisklasse (LOFI = Low Fidelity) und die komplette Zeitsteuerung erfolgt über Software. Hoffentlich wird es in Zukunft einmal Treiber für Windows geben, die auf bessere Hardware zugreifen. Es wären Synthesizer-Ansteuerungen, MIDI-Interface etc. denkbar. Das vorteilhafte an der Klangschnittstelle ist ihre Hardware-Unabhängigkeit. Heute ist sie noch auf »CGA-Qualität« beschränkt, morgen gibt es vielleicht »VGA-Qualität«. Für die Applikations-Programmierung hat das kaum Auswirkungen.

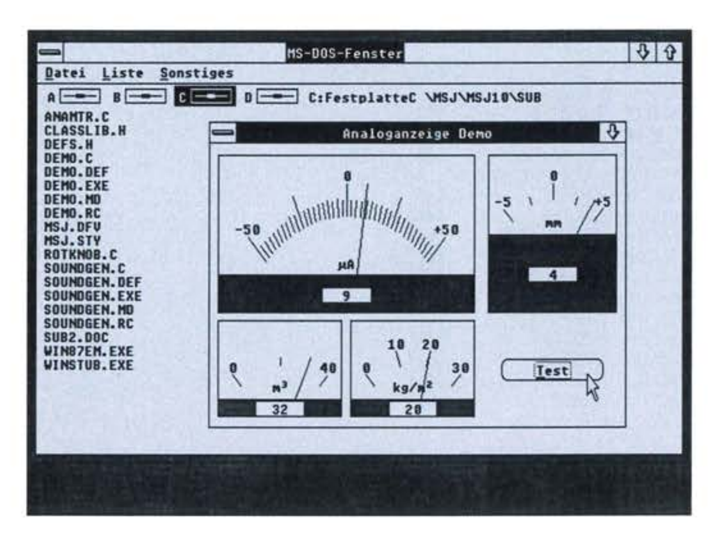


Bild 2: *Das Testprogramm DEMO.*

Soweit ein kurzer Einblick in das SOUND-Modul. Vielleicht werde ich in einem späteren Beitrag detaillierter einmal auf die Noten-Ansteuerung mit einer Applikation eingehen, die Peter Nortons BEEP-Programm ähnelt.

Übertragung einer Fensterklasse

Als Beispiel dafür, wie leicht eine Fensterklasse in eine andere Applikation übernommen werden kann, wurde ein Testprogramm für die Analoganzeige geschrieben, gezeigt in *Bild 2.* Die Fenster werden mit verschiedenen Skalen, Bereichen und Größen dargestellt, um die Implementierung zu testen. Durch Betätigen der Taste »Test« werden Zufallswerte an die vier Meßgeräte gesandt und angezeigt. Das Programm wurde hier aus Platzgründen nicht abgedruckt, es befindet sich jedoch inklusive des Quellcodes auf zu der zu diesem Heft erhältlichen Diskette.

Implementierung der neuen Fensterklassen

Nachdem die Arbeitsweise des Tongenerators klar geworden ist, sollten wir uns einmal ansehen, wie die neuen Fensterklassen implementiert worden sind. Schon die Länge der Listings zeigt, daß hierzu ziemlich viel Programmierarbeit vonnöten ist. Eine erfolgreiche Realisierung setzt ein umfangreicheres Verständnis der Windows-Grafikschnittstelle, des GDI voraus. Ich beschränke mich bei der Beschreibung auf grundlegende Techniken bei Fensterklassen-Implementierungen, das eigentliche Berechnen und Zeichnen der Skalen ist elementare Schul-Trigonometrie und wird nicht weiter erläutert. Wer sich dafür interessiert, muß die Listings Befehl für Befehl studieren.

Die Berechnungen wurden mit Fließkommaarithmetik vorgenommen. Dies ist weder schnell noch platzsparend. Bei einer professionellen Implementierung würde man für die Kreisberechnungen schnelle und kurze grafische Spezialalgorithmen auf Integer-Basis in Assembler verwenden. Dann wäre allerdings das Beispiel noch viel unverständlicher.

Der realisierte Kompromiß ist der Versuch des Optimums einer brauchbar schnellen Lösung einerseits, einem verständlichen didaktischen Beispiel andererseits und schließlich einem Listing, das von seiner Länge her nicht den Rahmen des *Microsoft System Journals* sprengt. Einige Abläufe in den Fensterklassenfunktionen waren ursprünlich didaktisch besser realisiert, wurden aber nachträglich »verkompliziert«, damit eine brauchbare Ausgabegeschwindigkeit erreicht wurde.

Viele grundlegende Dinge über Fensterklassen wurden von Kevin Welch bereits in der letzten Ausgabe des *Microsoft System Journals [3]* erwähnt, insbesondere die Organisation der lokalen statischen Variablen einer Fensterklasse. Diese Daten müssen für jedes Fenster einer Fensterklasse unabhängig zur Verfügung stehen. Als ersten Schritt würde man vielleicht jedem angelegten Fenster einen separaten Speicherblock zuweisen, in dem sich diese Daten befinden, und die Blöcke in einem Feld speichern. Beim Aufruf der Fensterfunktion müßte dann der entsprechende Block für das angesprochene Fenster ausgewählt werden. Und hier ergibt sich ein Problem. Der Fensterfunktion wird zwar eine eindeutige Nummer des Fensters übergeben, hierbei handelt es sich jedoch nicht um einen durchlaufenden Index, sondern um einen Bezug (*handle*), dessen Wert willkürlich von der Fensterverwaltung festgelegt wird. Er kann also nicht als Index in einem Feld verwendet werden. Die Zuordnung müßte über eine Hash-Funktion *Fensterbezug -> Tabellenindex* erfolgen, eine komplizierte und dennoch langsame Lösung.

Kevin Welch geht einen anderen Weg: Er speichert alle statischen Fensterdaten in dem Datenblock der Fensterdefinition, der mit `CreateWindow` für jedes Fenster einzeln angelegt wird. Man kann diesen Block am Ende um beliebige weitere Einträge vergrößern, indem man die Gesamtlänge dieser Einträge im Feld *cbWndExtra* der Fensterklassen-Definition WNDCLASS angibt (siehe hierzu auch *[2]*). Die Vorgehensweise hat jedoch eine kleine Einschränkung: Die zusätzlich angelegten Daten im Fensterdefinitionsblock sind nicht der Applikations-Speicherverwaltung direkt zugänglich; es besteht ausschließlich die Möglichkeit, über die Funktionen `GetWindowWord`, `SetWindowWord`, `GetWindowLong` und `SetWindowLong` auf diese Daten zuzugreifen. Statt der Adressierung einer Variablen ist jedesmal ein Funktionsaufruf nötig. In der Praxis wird man dies mit C-Makros realisieren. Dennoch ist die Lösung im Code platzintensiv und vor allem ziemlich langsam, besonders dann, wenn statische Variablen in Schleifen etc. verwendet werden müssen. Man kann auch nur auf WORD- und DWORD-Variablen zugreifen, lange Fließkommazahlen müssen über zwei aufeinanderfolgende Funktionsaufrufe gelesen oder geschrieben werden. Schließlich leidet auch das Verständnis und Wartung eines Programms bei Verwendung zahlreicher Makros ganz erheblich.

```
CC=CL >$*.ERR -c -AS -FPi -Gsw -Od -W2 -Zdp $*.C

SOUNDGEN.RES: SOUNDGEN.RC DEFS.H
 RC -r $*

SOUNDGEN.OBJ: SOUNDGEN.C DEFS.H CLASSLIB.H
 $(CC)

ROTKNOB.OBJ: ROTKNOB.C CLASSLIB.H
 $(CC)

ANAMTR.OBJ: ANAMTR.C CLASSLIB.H
 $(CC)

SOUNDGEN.EXE: SOUNDGEN.OBJ ROTKNOB.OBJ ANAMTR.OBJ\
              SOUNDGEN.RES SOUNDGEN.DEF
LINK4 SOUNDGEN+ROTKNOB+ANAMTR/A:16,/LI/M,,SLIBW+WIN87EM\
      +SLIBCEW/NOD,$*
 MAPSYM $*
 RC $*.RES
```

Listing 2: *Die MAKE-Datei SOUNDGEN.MD*

```
/*******************************************************************

 ----  S O U N D G E N  ----  MS-Windows Application ----------
                               Resource file

====================================================================

Copyright 1989 by Marcellus Buchheit, Buchheit software res.

Release 1.00 of 89-Mar-18 --- All rights reserved

*******************************************************************/

#include <WINDOWS.H>
#include "DEFS.H"

/* --------------------------
   dialog boxes specification
   --------------------------
*/

DBX_MAIN DIALOG LOADONCALL MOVEABLE DISCARDABLE
         0x8000,0,243,140
STYLE WS_OVERLAPPED|WS_CAPTION|WS_SYSMENU|WS_MINIMIZEBOX
CAPTION "Tongenerator"
BEGIN
 CONTROL "",IDMETER,"AnalogMeter",WS_CHILD,4,4,156,120
 CONTROL "r 1989 by Marcellus Buchheit, Karlsruhe",IDNONE,
         "static",SS_LEFT|WS_CHILD,4,128,156,8
 CONTROL "Bereich",IDNONE,
         "button",BS_GROUPBOX|WS_GROUP|WS_CHILD,168,6,68,68
 CONTROL "&Tief",IDRANGE10,"button",
         BS_RADIOBUTTON|WS_GROUP|WS_TABSTOP|WS_CHILD,
         172,18,40,12
 CONTROL "&Mittel",IDRANGE100,
         "button",BS_RADIOBUTTON|WS_CHILD,172,36,46,12
 CONTROL "&Hoch",IDRANGE1000,
         "button",BS_RADIOBUTTON|WS_CHILD,172,52,56,12
 CONTROL "Fein\t1\t\t\t4\t\t\t7\t\t\t10",IDFINE,"RotationKnob",
         WS_TABSTOP|WS_GROUP|WS_CHILD,172,80,60,60
END /* DBX_MAIN */
```

Listing 3: *Die Ressourcen-Datei SOUNDGEN.RC.*

```
/* dialog boxes */
#define DBX_MAIN         500
/* miscellaneous */
#define ICO_MAIN         900
/* dialog box entries */
#define IDNONE           0
#define IDMETER          2000
#define IDRANGE10        2001
#define IDRANGE100       2002
#define IDRANGE1000      2003
#define IDFINE           2004
#define IDMETER1         2010
#define IDMETER2         2011
#define IDMETER3         2012
#define IDMETER4         2013
#define IDTEST           2014
```

Listing 4: *Die Konstanten-Definitionsdatei DEFS.H.*

```
NAME SOUNDGEN
REALMODE
EXETYPE WINDOWS
DESCRIPTION 'Windows Sound Generator'
STUB 'WINSTUB.EXE'
CODE MOVABLE DISCARDABLE SHARED
DATA MOVABLE MULTIPLE
HEAPSIZE  4096
STACKSIZE 4096

EXPORTS
 fdMain @1
 fwAnalogMeter @2
 fwRotationKnob  @3
```

Listing 5: *Die Linker-Definitionsdatei SOUNDGEN.DEF.*

All dies hat mich dazu bewogen, mit einer weiteren Lösung zu experimentieren, mit der ich bis jetzt ganz gute Erfahrungen gemacht habe. Man verwendet für die lokalen statischen Variablen einer Fensterklasse einen separaten Speicherblock, den man beim Anlegen eines Fensters im lokalen Datenspeicher der Applikation erzeugt und beim Zerstören des Fensters wieder entfernt. Die Adressierung des Speicherblocks erfolgt jedoch nicht mehr über einen Index, sondern vielmehr über einen Speicherbezug. Dieser Speicherbezug (er ist natürlich für jedes Fenster der Klasse verschieden) wird in den Fensterdefinitionsblock der Fensterdefinition als einziges zusätzliches Datum abgelegt. Jedesmal beim Aufruf der Fensterfunktion wird mit `GetWindowWord` der Speicherbezug gelesen und die Adresse des Speicherblocks mit `LocalLock` ermittelt, bevor die `switch-Anweisung` die eingegangene Nachricht analysiert und bearbeitet. Über diese Adresse als Zeiger kann nun problemlos bei der Verarbeitung der Nachrichten innerhalb der Fensterfunktion auf die statischen lokalen Daten zugegriffen werden. Hierzu ist der Speicherblock als C-Record definiert und die statischen Variablen sind die Felder des Records. Statt beim Zugriff eine Funktion aufzurufen, erfolgt jetzt die Adressierung über einen festen Zeiger. Beim Einschalten der Optimierung des Microsoft-C-

Compilers wird der Zeiger wahrscheinlich im DI- oder SI-Register der CPU gehalten, der Zugriff erfolgt also sehr schnell.

Bevor die Fensterfunktion verlassen wird, muß der Speicherblock mit `LocalUnlock` wieder freigegeben werden. Da die Funktionen `LocalLock` und `LocalUnlock` die Anzahl der Sperrungen und Freigaben mitzählen, ist auch ein Reentrant-Aufruf der Fensterfunktion problemlos.

In den beiden realisierten eigenen Fensterklassen wird der Speicherblock beim Eingang der `WM_CREATE` angelegt und bei `WM_DESTROY` wieder entfernt. Ein Problem besteht jedoch noch: Die Funktion `LocalLock` funktioniert nur korrekt, wenn der angelegte Speicherblock auch tatsächlich vorhanden ist. Es gehen jedoch vor `WM_CREATE` und nach `WM_DESTROY` viele weitere Nachrichten ein, sodaß ein Aufruf von `LocalLock` fehlerhaft sein könnte. Man löst dieses Dilemma, indem man annimmt, daß ein Speicherblock nur funktioniert, wenn sein Bezug nicht NULL ist. Bei `WM_DESTROY` wird daher nach der Freigabe des Speicherblocks der Bezug im Fensterdefinitionsblock explizit auf `NULL` gesetzt. Und was passiert, bevor `WM_CREATE` übergeben wird? Nun, da der Speicherbezug sich im Fensterdefinitionsblock befindet, ist er am Anfang NULL, da die Funktion `CreateWindow` den Block komplett mit 0-Bytes initialisiert. Ich habe zwar nirgendwo in der Development-Kit-Beschreibung einen entsprechenden Hinweis darauf gefunden, aber zumindest Windows 2.0 führt die Initalisierung durch und ich gehe davon aus, daß dies bei zukünftigen Versionen auch so bleibt. (Eine über 25 Jahre alte Programmierer-Weisheit besagt, daß nicht-vorinitialisierte statische lokale Variablen wenig Sinn machen.)

```
/*
 * CLASSLIB.H --- Header file of the MS-Windows
 *                Window Classes Library
 * (C) 1989 by Marcellus Buchheit, Buchheit software research
 *
 * Release 1.0a of 89-Mar-18 --- All rights reserved
 */

/* --------------------
   Class "RotationKnob"
   --------------------
*/

/* control messages */
#define RK_SETRANGE (WM_USER)
#define RK_GETRANGE (WM_USER+1)
#define RK_SETPOS  (WM_USER+2)
#define RK_GETPOS  (WM_USER+3)

/* register-function */
BOOL RegisterRotationKnob(HANDLE hi);

/* following functions must be exported in *.DEF file:
   fwRotationKnob
*/

/* -------------------
   Class "AnalogMeter"
   -------------------
*/

/* control messages */
#define AM_UPDATE (WM_USER)
#define AM_RANGE  (WM_USER+1)

/* register-function */
BOOL RegisterAnalogMeter(HANDLE hi);

/* following functions must be exported in *.DEF file:
   fwAnalogMeter
*/
```

Listing 6: *Die Kopfdatei CLASSLIB.H der Klassenbibliothek.*

Implementierung des Drehknopfs

Listing 7 zeigt die Realisierung des Drehknopfs. Es sollen nur einige herausragende Merkmale beschrieben werden. Die Funktion `RegisterRotationKnob` registriert die Fensterklasse. Hier wird auch der Name der Fensterklasse angegeben. Er kann beliebig sein, sollte aber weder ein vordefinierter Fensterklassenname sein (etwa »ScrollBar«) noch mit irgendeinem anderen Fensterklassennamen kollidieren, der in der gleichen Applikation verwendet wird. Die Namen von Fensterklassen sind lokal zu Applikationen abgelegt. Zwei Applikationen mit gleichen Fensterklassennamen stören sich also nicht gegenseitig. Wird jedoch ein Fensterklassenname von `CreateWindow` nicht in der eigenen Applikation gefunden, wird auch in anderen Modulen nach diesem Namen gesucht. Dadurch ist auch der Zugriff auf System-Fensterklassen wie »ScrollBar« möglich.

Die einzige Besonderheit in der Registrierungsfunktion ist, daß dem Feld *cbWndExtra* nicht 0, sondern die Anzahl der zusätzlich benötigten Bytes innerhalb des Fensterdefinitionsblocks angegeben wird. Da wir diese Daten explizit mit `LocalAlloc` anlegen, genügt es, Platz für einen Speicherbezug (Typ `HANDLE`) zu schaffen.

Der Drehknopf zeichnet sich abhängig von der Größe, die ihm bei `CreateWindow` beziehungsweise in der Dialogfelddefinition zugewiesen wurde. Im letzteren Fall ist die Größe relativ zur Zeichengröße des SYSTEM-Zeichensatzes am Bildschirm. Bei `CreateWindow` sollte eine solche Größenrelation bei der Implementierung berücksichtigt werden, damit nicht der Drehknopf bei CGA zu groß und bei 1024x768 Pixel pro Bildschirm zu winzig ist. Die Größe wird mit der Nachricht `WM_SIZE` der Fensterfunktion `fwRotationKnob` übergeben und viele statische Variablen sind vorberechnete Größenangaben für bestimmte Teile des Knopfs und der Skala.

Für bestimmte Größenberechnungen benötigt man in der Fensterfunktion auch die Größe der Bildschirmzeichen. Leider läßt sich diese nicht einfach über die Funktion `GetSystemMetrics` ermitteln. Man muß vielmehr mit `CreateIC` einen Displaykontext schaffen, um die Zeichenhöhe und -breite über die Funktion `GetTextMetrics` zu erhalten. Da der Wert jedoch für alle Zeichnungen konstant ist, können wir ihn in einer gemeinsamen statischen Variable ablegen und brauchen ihn nur einmal zu berechnen.

Wird der Eingabefokus dem Knopf zugewiesen, muß die Oberfläche grau gezeichnet werden. Dies geschieht jedoch nicht bei WM_SETFOCUS direkt, sondern, wie auch bei Applikationsfenstern üblich erfolgen alle Zeichenoperationen zentral bei WM_PAINT. WM_SETFOCUS invalidiert lediglich das Rechteck des eigentlichen Knopfs (die Skala braucht ja nicht neu gezeichnet zu werden). Dagegen muß der gesamte Knopf einschließlich Skala neu gezeichnet werden, falls mit WM_SETTEXT die Skala geändert wird.

Ein allgemeines Problem beim Zeichnen von Objekten mit GDI ist, daß nach Erzeugen eines Zeichenkontextes (*Display Context*) als Zeichenfarbe schwarz und weiß und nicht etwa die aktuell eingestellten Fenstervordergrund- und Fensterhintergrundfarben gesetzt sind. Wird dies nicht berücksichtigt, würden die gezeichneten Objekte sehr merkwürdig aussehen, falls ein Benutzer gerne schwarze Dialogfelder mit weißer Schrift hat (zum Beispiel weil sein Monitor stark flimmert). Deshalb wird nach dem BeginPaint-Aufruf beim Bearbeiten der WM_PAINT-Nachricht zunächst die aktuelle Vordergrund- und Hintergrundfarbe bestimmt und gesetzt. Jetzt wird der Knopf gezeichnet. Er wird mit einem hellgrauen Standardmuster gefüllt, falls der Eingabefokus ihm zugewiesen wurde.

Anschließend wird die Skala gezeichnet. Das Zeichnen von Bildschirmteilen erfolgt bei Windows bekanntlich immer nur dann, wenn sie ungültig wurden, indem sie bei InvalidateRect oder ähnlichen Funktionen angegeben wurden. Dies gilt jedoch nur für die eigentlichen Zeichenoperationen. Die zeitaufwendigen Berechnungen der Skala würden trotzdem durchgeführt. Deshalb wird vor dem Zeichnen der Skala ermittelt, ob der neuzuzeichnende Fensterausschnitt die Skala betrifft. Dies kann über das Rechteck ermittelt werden, das im Feld *rcPaint* der von BeginPaint beschriebenen PAINTSTRUCT-Struktur steht.

Beim Zeichnen der Skala wird zunächst die Anzahl der Skalenstriche ermittelt und daraus durch einfache Kreisberechnungen die Positionen der einzelnen Striche. Beim Zeichnen der Striche wird gleichzeitig der Skalentext positioniert. Dies ist abhängig vom Winkel des Strichs. Glücklicherweise gibt es die Funktion SetTextAlign, die diese Aufgabe übernimmt. Schließlich wird der Anzeigestrich auf dem Knopf neu gezeichnet werden. Dies geschieht jedesmal, wenn der Knopf gedreht wird.

Am Schluß der Fensterfunktion werden schließlich die hinzugekommenen privaten Klassen-Nachrichten bearbeitet. Hier werden im wesentlichen die übergebenen Werte statischen Variablen zugewiesen beziehungsweise die früher gesetzten Werte zurückgegeben.

Die Maus- und Tastaturbehandlung des Drehknopfs

Wäre der Knopf ein Ausgabe-Dialogobjekt, wären wir jetzt fertig. Aber bei Eingabe-Dialogobjekten müssen noch Mausbewegungen und Tastendrücke auf der Tastatur oder

```
/************************************************************

 ----  R O T K N O B  ------ MS-Windows Class Module ---------

=============================================================

This MS-Windows module contains a window class which realizes
a rotation knob to select analog values as alternative of a
scroll bar.

-------------------------------------------------------------

Copyright 1989 by Marcellus Buchheit, Buchheit software res.

Release 1.00 of 89-Mar-19 --- All rights reserved.

*************************************************************/

/* define/undefine non-debugging option name */
#undef  NDEBUG

#define  NOMINMAX
#include <WINDOWS.H>
#include "CLASSLIB.H"

/* ANSI-C headers */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/* window function parameter macros */
#define P2LO LOWORD(ulP2)
#define P2HI HIWORD(ulP2)

/* local module data values */
static WORD sxChar; /* width of a SYSTEM character */
static WORD syChar; /* height of a standard SYSTEM character */

/* ------------------------------------------------
   structure which contains the local windows data
   ------------------------------------------------
*/

typedef struct
 {RECT wrcKnob; /* rectangle of knob (without scale) */
  HRGN hrgKnob; /* circle range of knob */
  int xCenter,yCenter; /* center location of knob */
  WORD sScale; /* number of entries in scale */
  double vfR1,vfR2; /* inner/outer radius of scale */
  WORD uMinVal,uMaxVal; /* scale range minimum/maximum */
  WORD uCurValue; /* current selected scale value */
  WORD uTrackStart; /* track location
                       (where mouse button pressed) */
  WORD uTrackLoc; /* current track location
                     (during mouse button pressed) */
  BOOL bTrackKnob; /* flag "during tracking"
                      (mouse button pressed) */
  double vfTrackAngle,vfLastAngle; /* current/last angle during
                                      tracking */
 }
WNDDATA;

/************************************************************
 f w R o t a t e K n o b
=============================================================

/// window function ///

This function processes any messages received by a window
of the window class "RotationKnob".

*************************************************************/


LONG FAR PASCAL fwRotationKnob
                (HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)
```

Listing 7: *Die Datei ROTKNOB.C (RotationKnob).*

```
{WNDDATA *r; /* pointer to local window data */
 LONG vRet; /* return value */
 HANDLE hData; /* local memory handle of window class data */
 int xStart,yStart,xStop,yStop,sx,sy,sv;
 LOCALHANDLE hmText;
 LONG vl;
 double vfAngle,vfDelta,vfCos,vfY;
 double vfR1Sqr,vfR2Sqr;
 PSTR rzText,rz;
 PAINTSTRUCT wps;
 RECT wrc;
 HDC hdc;
 HPEN hpnDraw;
 HBRUSH hbrBkgr;
 int s,i,iCmd;
 #define PI 3.14159265354L

 /* ----- get pointer to window data block ----- */
 hData=GetWindowWord(hw,0);
 if (hData) r=(WNDDATA*)LocalLock(hData);
 vRet=0L;
 /* ----- analyse message ----- */
 switch (iMsg)
  {case WM_CREATE:
    /* allocate memory block for local memory */
    hData=LocalAlloc
     (LMEM_MOVEABLE|LMEM_ZEROINIT,sizeof(WNDDATA));
    SetWindowWord(hw,0,hData); r=(WNDDATA*)LocalLock(hData);
    if (sxChar==0)
     {/* determine height and size of console characters */
      TEXTMETRIC wtm;
      hdc=CreateIC("Display",NULL,NULL,NULL);
      GetTextMetrics(hdc,&wtm);
      sxChar=wtm.tmAveCharWidth; syChar=wtm.tmHeight;
      DeleteDC(hdc);
     } /* if */
    goto Return; /* return 0L */
   case WM_GETDLGCODE:
    /* if focus: window wants direction keys */
    if (hw==GetFocus()) vRet=DLGC_WANTARROWS;
    goto Return; /* return 0L */

   case WM_SIZE:
    /* determine center of knob: in middle of window */
    r->xCenter=P2LO/2; r->yCenter=P2HI/2;
    s=(P2HI-4*syChar)/2; /* radius of circle */
    r->wrcKnob.left=r->xCenter-s;
    r->wrcKnob.top=r->yCenter-s;
    r->wrcKnob.right=r->xCenter+s;
    r->wrcKnob.bottom=r->yCenter+s;
    r->hrgKnob=CreateEllipticRgnIndirect(&r->wrcKnob);
    /* determine scale size: outer & inner radius */
    r->vfR1=(double)((r->wrcKnob.right-r->wrcKnob.left)/2+2);
    r->vfR2=1.5*r->vfR1; /* length of scale line:
                            half the radius */
    goto Return; /* return 0L */

   case WM_SETFOCUS:
   case WM_KILLFOCUS:
    /* redraw inside of knob (gray if active focus) */
    InvalidateRect(hw,NULL,FALSE);
    break; /* not consumed */
   case WM_SETTEXT:
    /* redraw full window */
    InvalidateRect(hw,NULL,FALSE); UpdateWindow(hw);
    break; /* not consumed */
   case WM_PAINT:
    hdc=BeginPaint(hw,&wps);
    hpnDraw=CreatePen
     (PS_SOLID,1,GetSysColor(COLOR_WINDOWTEXT));
    hbrBkgr=CreateSolidBrush(GetSysColor(COLOR_WINDOW));
    SelectObject(hdc,hpnDraw);
    SelectObject
     (hdc,hw==GetFocus()?
      GetStockObject(LTGRAY_BRUSH):hbrBkgr);
    Ellipse
     (hdc,r->wrcKnob.left,r->wrcKnob.top,r->wrcKnob.right,
      r->wrcKnob.bottom);
```

Listing 7: *(Fortsetzung)*

```
    if (!EqualRect(&wps.rcPaint,&r->wrcKnob))
     {/* redraw scale of knob */
      SelectObject(hdc,hbrBkgr);
      SetTextColor(hdc,GetSysColor(COLOR_WINDOWTEXT));
      SetBkMode(hdc,TRANSPARENT);
      s=GetWindowTextLength(hw);
      hmText=LocalAlloc(LMEM_MOVEABLE,s+1);
      rzText=LocalLock(hmText); GetWindowText(hw,rzText,s+1);
      /* determine number of scale points */
      for (rz=rzText,r->sScale=0;*rz++;)
       {if (*rz=='\t') r->sScale++;
       } /* for */
      /* --- draw text of knob name (at bottom) --- */
      rz=rzText;
      while (rz!=0&&*rz!='\t') rz++; /* determine length of
                                         name */
      SetTextAlign(hdc,TA_CENTER|TA_BOTTOM);
      GetClientRect(hw,&wrc);
      TextOut(hdc,wrc.right/2,wrc.bottom-2,rzText,rz-rzText);
      /* --- draw scale lines and text --- */
      vfR1Sqr=r->vfR1*r->vfR1; vfR2Sqr=r->vfR2*r->vfR2;
      vfDelta=3.0*PI/2.0/(r->sScale-1); /* 3/2 PI is
                                           270 degree */
      /* calculate scale points for right half */
      for (i=0,vfAngle=-3.0*PI/4.0;i<r->sScale;
           vfAngle+=vfDelta,i++)
       {vfCos=cos(vfAngle);
        vfY=vfCos*r->vfR1; yStart=r->yCenter-(int)vfY;
        xStart=r->xCenter+
              (int)sqrt(vfR1Sqr-vfY*vfY)*(vfAngle<0.0?-1:+1);
        vfY=vfCos*r->vfR2; yStop=r->yCenter-(int)vfY;
        xStop=r->xCenter+
              (int)sqrt(vfR2Sqr-vfY*vfY)*(vfAngle<0.0?-1:+1);
        MoveTo(hdc,xStart,yStart); LineTo(hdc,xStop,yStop);

        if (*rz)
         {/* print text to scale point */
          rzText=++rz; while (*rz&&*rz!='\t') rz++;
          if (*rzText!='\t' && *rzText!='\0')
           {/* no empty text: draw it */
            SetTextAlign
             (hdc,2*i>=r->sScale? TA_BASELINE|TA_LEFT:
               2*i==r->sScale-1?
               TA_CENTER|TA_BOTTOM: TA_BASELINE|TA_RIGHT
             );
            TextOut
             (hdc,xStop,yStop,rzText,rz-rzText
             );
           } /* if */
         } /* if */
       } /* for */
      /* delete created brush, set default again */
      LocalUnlock(hmText); LocalFree(hmText);
     } /* if */
    /* repaint interior of knob (scale line) */
    SelectObject(hdc,r->hrgKnob);
    vfY=r->vfR1*cos
     (vfAngle=3.0*PI/2*
      ((double)(r->uCurValue-r->uMinVal)/
      (double)(r->uMaxVal-r->uMinVal)-0.5)
     );
    yStop=r->yCenter-(int)vfY;
    xStop=r->xCenter+
     (int)sqrt(r->vfR1*r->vfR1-vfY*vfY)*(vfAngle<0.0?-1:+1);
    MoveTo(hdc,r->xCenter,r->yCenter);
    LineTo(hdc,xStop,yStop);
    EndPaint(hw,&wps);
    DeleteObject(hpnDraw); DeleteObject(hbrBkgr);
    goto Return; /* return 0L */

   case WM_LBUTTONDOWN:
    /* --- handle knob via mouse --- */
    SetFocus(hw); /* focus to knob */
    if (PtInRegion(r->hrgKnob,P2LO,P2HI))
     {/* location inside knob: start knob tracking */
      r->bTrackKnob=TRUE; /* knob is rotated */
      r->uTrackStart=r->uCurValue; /* set tracking start */
      r->uTrackLoc=r->uCurValue; /* init track value */
```

Listing 7: *(Fortsetzung)*

```
      r->vfTrackAngle=atan2
        ((double)((int)P2LO-r->xCenter),
         (double)(r->yCenter-(int)P2HI));
     }
    else
     {/* location outside knob: new location if in scale */
      sx=(int)P2LO-r->xCenter; sy=r->yCenter-(int)P2HI;
      sv=r->uMaxVal-r->uMinVal;
      vfDelta=
       sqrt((double)((long)sx*(long)sx+(long)sy*(long)sy));
      if (vfDelta<r->vfR1||vfDelta>r->vfR2)
         break; /* out of scale */
      vl=(int)((atan2((double)sx,(double)sy)/(3.0/2.0*PI))*
         (double)sv);
      vl+=(r->uMinVal+r->uMaxVal)/2; /* into range
                                       uMinVal..uMaxVal */
      /* angle too far from left/right scale end => break */
      if (vl<r->uMinVal-(sv+5)/10||vl>r->uMaxVal+(sv+5)/10)
         break;
      /* set new knob location */
      r->uCurValue=(vl<(LONG)r->uMinVal)?r->uMinVal:
                (vl>(LONG)r->uMaxVal)?r->uMaxVal:(int)vl;
      /* send message/new location (user must update knob) */
      SendMessage
       (GetParent(hw),WM_VSCROLL,SB_THUMBPOSITION,
        MAKELONG(r->uCurValue,hw));
     } /* if */
    break;
  case WM_MOUSEMOVE:
  case WM_LBUTTONUP:
    if (r->bTrackKnob)
     {if (PtInRegion(r->hrgKnob,P2LO,P2HI))
       {/* mouse within knob: determine new location */
        vfAngle=atan2((double)((int)P2LO-r->xCenter),
                      (double)(r->yCenter-(int)P2HI))
          -r->vfTrackAngle;
        if (fabs(vfAngle-r->vfLastAngle)<3.0/2.0*PI)
         {/* rotation angle doesn't overwind knob: new loc.*/
          r->vfLastAngle=vfAngle;
          vl=r->uTrackLoc+(int)(vfAngle/(3.0/2.0*PI)*
             (double)(r->uMaxVal-r->uMinVal));
          if (vl>r->uMaxVal) vl=r->uMaxVal;
          if (vl<r->uMinVal) vl=r->uMinVal;
          if (iMsg!=WM_LBUTTONUP)
           {/* send message about new location during mouse
               button down */
            SendMessage
             (GetParent(hw),WM_VSCROLL,SB_THUMBTRACK,
              MAKELONG((int)vl,hw)
             );
           } /* if */
          /* repaint knob rectangle immediately */
          InvalidateRect(hw,&r->wrcKnob,FALSE);
          UpdateWindow(hw);
         }
       }
      else
       {/* outside knob: reset location before tracking */
        if (r->uCurValue!=r->uTrackStart)
         {r->uCurValue=r->uTrackStart; /* set new value */
          /* repaint knob rectangle immediately */
          InvalidateRect(hw,&r->wrcKnob,FALSE);
          UpdateWindow(hw);
         } /* if */
       } /* if */
      if (iMsg==WM_LBUTTONUP)
       {/* end of tracking mode: send last location */
        r->bTrackKnob=FALSE;
        SendMessage
         (GetParent(hw),WM_VSCROLL,SB_THUMBPOSITION,
          MAKELONG(r->uCurValue,hw)
         );
       } /* if */
      goto Return; /* return 0L */
     } /* if */
    break;
  case WM_KEYDOWN:
    /* --- handle knob via keyboard --- */
```

Listing 7: *(Fortsetzung)*

```
      switch (uP1)
       {case VK_RIGHT:
        case VK_DOWN:
          iCmd=SB_LINEDOWN; break;
        case VK_LEFT:
        case VK_UP:
          iCmd=SB_LINEUP; break;
        case VK_NEXT:
          iCmd=SB_PAGEDOWN; break;
        case VK_PRIOR:
          iCmd=SB_PAGEUP; break;
        case VK_HOME:
          iCmd=SB_TOP; break;
        case VK_END:
          iCmd=SB_BOTTOM; break;
        default:
          goto DefWin; /* not consumed */
       } /* switch */
      SendMessage
       (GetParent(hw),WM_VSCROLL,iCmd,MAKELONG(0,hw));
      goto Return; /* return 0L, key is consumed */
    case RK_SETRANGE:
      /* set new value in new range */
      r->uMinVal=P2LO; r->uMaxVal=P2HI;
      r->uCurValue=uP1;
      /* repaint full window */
      InvalidateRect(hw,NULL,TRUE); UpdateWindow(hw);
      goto Return; /* return 0L */
    case RK_GETRANGE:
      /* return current selected range */
      vRet=MAKELONG(r->uMinVal,r->uMaxVal);
      goto Return;
    case RK_SETPOS:
      r->uCurValue=uP1; /* set new value */
      /* repaint knob rectangle */
      InvalidateRect(hw,&r->wrcKnob,FALSE); UpdateWindow(hw);
      goto Return; /* return 0L */
    case RK_GETPOS:
      /* return current knob position */
      vRet=r->uCurValue;
      goto Return;
    case WM_DESTROY:
      /* destroy knob region */
      DeleteObject(r->hrgKnob);
      /* destroy window data memory block, invalidate handle */
      LocalUnlock(hData); LocalFree(hData);
      SetWindowWord(hw,0,NULL);
      return 0L;
   } /* switch */
  DefWin:
  vRet=DefWindowProc(hw,iMsg,uP1,ulP2);
  /* return from function: unlock window data memory block */
  Return:
  if (hData) LocalUnlock(hData);
  return vRet;
 } /* fwRotationKnob() */


/*****************************************************************
 R e g i s t e r R o t a t i o n K n o b
=================================================================

This function registers the window class "RotationKnob".
*****************************************************************/

BOOL RegisterRotationKnob(HANDLE hi)
 {WNDCLASS wwc;
  wwc.lpszClassName="RotationKnob";
  wwc.hInstance=hi;
  wwc.lpfnWndProc=fwRotationKnob;
  wwc.hCursor=LoadCursor(NULL,IDC_ARROW);
  wwc.hbrBackground=COLOR_WINDOW+1;
  wwc.style=CS_HREDRAW|CS_VREDRAW;
  wwc.hIcon=NULL; wwc.lpszMenuName=NULL;
  wwc.cbClsExtra=0; wwc.cbWndExtra=sizeof(HANDLE);
  /* register window, return status */
  return RegisterClass(&wwc);
 } /* RegisterRotationKnob() */
```

Listing 7: *(Ende)*

der Maus verarbeitet werden. Bei einem Druck der linken Maustaste muß zunächst festgestellt werden, ob sie innerhalb des Knopfs oder innerhalb der Skala gedrückt worden ist. Hierzu wurde bereits beim `WM_SIZE` eine Kreisregion des Knopfs angelegt und die nützliche Funktion `PtInRegion` entlastet von komplizierten Punktzuordnungsoperationen. Liegt der Punkt auf der Skala, ist die weitere Vorgehensweise relativ einfach: Aus dem Punkt wird der Winkel ermittelt (hierzu habe ich die etwas genauere und schnellere `atan2`-Funktion verwendet), aus diesem die neue Position und mit Hilfe von `SendMessage` letztere an das Vaterfenster übergeben. Alle Mitteilungsnachrichten von Dialogobjekten müssen an das entsprechende Vaterfenster übergeben werden, das beispielsweise ein Dialogfeld ist und die Nachrichten an seine Dialogfeld-Funktion weiterleitet, wo sie (wie bei SOUNDGEN) verarbeitet werden. Besondere Toleranzen wurden eingeführt, um etwas laxer den Anfang oder das Ende der Skala anklicken zu können.

Komplizierter ist die Verarbeitung, wenn der Knopf mit der Maus »gezogen« werden soll, nachdem das Innere des Knopfs angeklickt wird. Hierzu muß zunächst ein Zustand gesetzt werden, daß der Knopf gezogen werden soll. Dies erfolgt in der Variable `bTrackKnob`. Die aktuelle Position des Drehknopfs wird als Ausgangspunkt gesetzt, da die weiteren Positionieren mit einem relativen Winkel dazu erfolgen. Bei jedem Eingang der Nachricht `WM_MOUSEMOVE` wird die neue Position errechnet und dem Vaterfenster mitgeteilt. Beim Loslassen der Maustaste wird schließlich die endgültige Position noch einmal dem Vaterfenster übergeben. Dies ist kompatibel zu den Rolleisten.

Besonders sorgfältig müssen »Bedienungsfehler« behandelt werden. Dies ist beispielsweise ein Verlassen des Knopfbereichs mit dem Mauszeiger beim Nachziehen des Knopfs oder das Überdrehen des Knopfs. Hier müssen gegebenenfalls an der jetzigen Implementierungen noch kleine Verbesserungen erfolgen.

Bei Windows sollte alles auch ohne Maus bedienbar sein, nicht nur, weil es immer noch Benutzer gibt, die keine Maus besitzen (man kann sie nur bedauern), sondern weil auch verschiedene Sicherheitsanforderungen eine Systembenutzung ohne Maus vorschreiben könnten oder zentrale Makrosteuerungen unter Windows leichter mit Tastaturbefehlen zu programmieren sind. Daher besitzt der Drehknopf auch eine komplette Tastaturschnittstelle, die kompatibel zur Rolleiste ist. Es werden die brauchbaren Tasten beim Eingang der `WM_ KEYDOWN`-Nachricht weiterverarbeitet und in ensprechende `WM_VSCROLL`-Mitteilungsfunktionen `umgewandelt`. Eigentlich sollte es bei der ganzen Tastaturansteuerung keine Probleme geben.

Wie ein Dialogelement Steuertasten erhält

Ein sehr großes Problem hatte ich jedoch vor etwa einem Jahr, als ich meine erste eigene Eingabe-Fensterklasse entwickelte. Es handelte sich hierbei um ein Laufwerk-Aus-

```
/*************************************************************

 ---- A N A M T R -------- MS-Windows Class Module ---------

=============================================================

This MS-Windows module contains a window class which realizes
an analog meter to display values as alternative to a digital
output.

-------------------------------------------------------------

Copyright 1989 by Marcellus Buchheit, Buchheit software res.

Release 1.00 of 89-Mar-19 --- All rights reserved.

*************************************************************/

/* define/undefine non-debugging option name */
#undef  NDEBUG

#define NOMINMAX
#include <WINDOWS.H>
#include "DEFS.H"
#include "CLASSLIB.H"

/* ANSI-C headers */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/* window function parameter macros */
#define P2LO LOWORD(ulP2)
#define P2HI HIWORD(ulP2)

/* local module data values */
static WORD sxChar; /* width of a SYSTEM character */
static WORD syChar; /* height of a standard SYSTEM character */


/* ---------------------------------------------
   structure which contains the local windows data
   ---------------------------------------------
*/

typedef struct
 {HWND hwDigital; /* window of digital display value */
  LOCALHANDLE hmText; /* handle to window's string block */
  int syScale; /* vertical size of scale */
  int sScale; /* number of lines in scale */
  int sxClient,syClient; /* size of window's client area */
  int uMinVal,uMaxVal; /* minimum/maximum values for scale */
  double vfR3,vfR3Sqr; /* radius & square of radius of scale */
  int uScaleDelta; /* difference between name and no-name
                      scale line */
  struct
   {int xStart;
    int yStart;
    int xStop;
    int yStop;
   }
  ScaleTable[41]; /* locations of the scale lines */
  int vxScale,vyScale; /* scale location of analog pointer */
  int yScaleCenter;  /* center of scale and anlog pointer */
  int sxHalf; /* half size of instrument */
 }
WNDDATA;


/*************************************************************
 f w A n a l o g M e t e r
=============================================================

### window function ###
This function processes any messages received by a window of
the window class "AnalogMeter".

*************************************************************/
```

Listing 8: *Die Datei ANAMTR.C (»AnalogMeter«).*

```
LONG FAR PASCAL fwAnalogMeter
                (HWND hw,WORD iMsg,WORD uP1,DWORD ulP2)

 {WNDDATA *r; /* pointer to local window data */
  LONG vRet; /* return value */
  HANDLE hData; /* local memory handle of window class data */
  PAINTSTRUCT wps;
  HDC hdc;
  HPEN hpnDraw;
  HBRUSH hbrBkgr;
  int iTab1,iTab2,i,s;
  double vfR1,vfR1Sqr,vfR2,vfR2Sqr;
  double vfAngle,vfDelta,vfCos;
  double vfY1,vfY2,vfY3;
  int vX1,vX2,vX3,vY1,vY2,vY3;
  LPSTR rzText,rz;
  char z[6];
  RECT wrc;
  #define PI 3.14159265354L

  /* ----- get pointer to window data block ----- */
  hData=GetWindowWord(hw,Ø);
  if (hData) r=(WNDDATA*)LocalLock(hData);
  vRet=ØL;
  /* ----- analyse message ----- */
  switch (iMsg)
   {case WM_CREATE:
      /* allocate memory block for local memory */
      hData=LocalAlloc
       (LMEM_MOVEABLE|LMEM_ZEROINIT,sizeof(WNDDATA));
      SetWindowWord(hw,Ø,hData); r=(WNDDATA*)LocalLock(hData);
      r->sScale=2; r->uScaleDelta=32767;
      if (sxChar==Ø)
       {/* determine height and size of console characters */
        TEXTMETRIC wtm;
        hdc=CreateIC("Display",NULL,NULL,NULL);
        GetTextMetrics(hdc,&wtm);
        sxChar=wtm.tmAveCharWidth; syChar=wtm.tmHeight;
        DeleteDC(hdc);
       } /* if */
      /* complete window style */
      SetWindowLong
       (hw,GWL_STYLE,
        GetWindowLong(hw,GWL_STYLE)|WS_CLIPCHILDREN);
      /* create digital display */
      r->hwDigital=CreateWindow
       ("static","",WS_BORDER|SS_CENTER|WS_VISIBLE|WS_CHILD,
        Ø,Ø,Ø,Ø,hw,NULL,((CREATESTRUCT*)ulP2)->hInstance,NULL
       );
      /* allocate memory for window text */
      r->hmText=LocalAlloc(LMEM_MOVEABLE|LMEM_ZEROINIT,1);
      goto Return; /* return ØL */
    case AM_RANGE:
      /* set only parameters, no repaint */
      r->uMinVal=(int)P2LO; r->uMaxVal=(int)P2HI;
      r->sScale=uP1; goto Return; /* return ØL */
    case WM_SETTEXT:
      rzText=(LPSTR)ulP2;
      /* determine size incl \Ø */
      rz=rzText; while (*rz++); s=rz-rzText;
      r->hmText=LocalReAlloc
       (r->hmText,LMEM_MOVEABLE|LMEM_ZEROINIT,s);
      /* transfer string into buffer */
      rz=LocalLock(r->hmText); while (*rz++=*rzText++);
      LocalUnlock(r->hmText);
      goto NewScale; /* paint new scale */
    case WM_SIZE:
      /* store size of client rectangle */
      r->sxClient=P2LO; r->syClient=P2HI;
      r->sxHalf=r->sxClient/2;
   NewScale:
      /* determine number of longer scale lines
         (first part is unity) */
      rz=LocalLock(r->hmText);
      for (s=-1;*rz;s++)
       {/* skip characters until next tabulate */
        while (*rz && *rz++!='\t');
       } /* for */
```

Listing 8: *(Fortsetzung)*

wahlfenster, ähnlich dem im MS-DOS-Fenster. Es sollte über Maus und die Cursor-Richtungstasten zu bedienen sein. Aber es stellte sich heraus, daß Dialogfelder die Richtungstasten für das eigene Positionieren verwenden (beispielsweise die Tabulator-Taste für das Wandern von Element zu Element) und überhaupt nicht an ein Objekt-Tochterfenster weitergeben. Dies steht auch in der Development-Kit-Literatur, doch das heißt es lapidar: »If a dialog box has scroll bars, the application must provide an appropriate keyboard interface for the scroll bars« (Kapitel 2.2.7.9). Doch wie sollte das geschehen? Immerhin wußte ich ja, daß die Rolleiste die Cursortasten korrekt verarbeitete. Eine Analyse der Rolleisten-Fensterfunktion mit Hilfe des Debuggers scheiterte aufgrund der Komplexität der Funktion. Immerhin bekam ich heraus, daß die Rolleiste die Cursortaste geschickt bekam. Aber wie? Nach einigem erfolglosem Suchen entschloß ich mich, die geeigneten Tasten über den Systemeingriff `WH_MSGFILTER` der Funktion `SetWindowsHook` abzufangen und an das Fenster zu schicken. Nach einigen Versuchen klappte dies ganz ausgezeichnet, aber das Ergebnis war fürchterlich kompliziert und umständlich. Erst als ich einige Wochen später ein ganz anderes Problem hatte und für dessen Lösung die Liste der Nachrichten Schritt für Schritt alphabetisch durchging, entdeckte ich die Nachricht `WM_GETDLGCODE`. Und die war extra dafür geschrieben worden, um mein Problem zu lösen. Sie wird jedesmal an ein Objekt innerhalb eines Dialogfelds geschickt, sobald dieses den Eingabefokus erhält, damit die Feldverwaltung gesagt bekommt, welche Tasten sie weiterhin selbst interpretieren soll und welche sie an das Tochterfenster schicken muß. Die Lösung meines Problems auf höchst einfache Weise! Und eine solche nützliche Nachricht wurde weder beim obigen Zitat noch in der Übersicht der Nachrichten mit irgendeinem Wort erwähnt.

Ich erzähle dies nur deshalb hier, weil sicherlich der eine oder andere Leser bei einem Problem mit Windows nicht weiterkommt. Da leider in Deutschland immer noch keine Hotline nach Amerika besteht, bleibt nur die Möglichkeit, in der Dokumentation die Lösung des Problems »linear« zu suchen, indem man die Funktionen und die Nachrichten alphabetisch durchgeht. Diese beiden Kapitel sind die einzig vollständigen der Devolpment-Kit-Dokumentation. Die Vorgehensweise ist zwar sehr zeitraubend, führt aber letztendlich zum Ziel und man lernt (wie in meinem Beispiel) eine ganze Menge über Dinge hinzu, die man eigentlich gar nicht gesucht hatte aber sicherlich irgendwann einmal braucht.

Implementierung des Analoganzeige-Geräts

Listing 8 zeigt, wie das Analoganzeige-Gerät programmiert wurde. Die Skala wird ähnlich wie beim Drehknopf zunächst berechnet und dann gezeichnet. Da jedoch der Analogzeiger über die Skala wandert (er würde sonst zu verkrüppelt aussehen), muß diese bei jedem Wertwechsel

zumindest teilweise neu gezeichnet werden. Um dann die zeitaufwendigen Kreisberechnung zu vermeiden, wird die Skala bereits beim `WM_SIZE` berechnet und in Form von Strichpositionen in dem Feld `ScaleTable` abgelegt. Beim Zeichnen der Skala nach dem Eintreffen der `WM_PAINT`-Nachricht kann dann auf diese Tabelle zurückgegriffen werden. Das Neuzeichnen des Zeigers erfolgt aus Geschwindigkeitsgründen ausnahmesweise nicht nach `WM_PAINT`, sondern bereits beim Bearbeiten der Nachricht `WM_UPDATE`. Es wird zunächst der alte Zeiger mit der Hintergrundfarbe überschrieben und dann der neue Zeiger gezeichnet. Es muß jedoch noch die Skala und auch die Skalenbeschriftung teilweise erneuert werden. Der hierfür neu zu zeichnende Teil wird als Rechteck aus dem alten und dem neuen Skalenwert ermittelt. Bei kleinen Wertänderungen muß also nur ein kleiner Bereich der Skala neu gezeichnet werden.

Ein weiteres Problem betraf die graue »Abdeckung« des Meßgeräts. Schließlich soll in ihr nicht der Zeiger zu sehen sein. Deshalb wird mit der Funktion `IntersectClipRect` vor dem Zeichnen des Zeigers der Zeichenbereich auf die Skala des Meßgeräts begrenzt. Solche nützlichen Funktionen bewirken immer wieder gewisse Glückserlebnisse, vorausgesetzt man kennt sie!

Ursprünglich hatte ich vorgehabt, das Meßgerät mit einem eigenen Rahmen zu zeichnen, indem die `WM_NCxxx`-Nachrichten interpretiert werden und der Zeichenbereich des Fensters (*Client area*) auf die Skala begrenzt wird. Aber die Dokumentation war in dem Punkt dermaßen verworren, daß ich davon zunächst wieder Abstand genommen habe. Vermutlich kann sich bei Microsoft niemand vorstellen, daß man Tochterfenster mit eigenem Rahmen brauchen könnte.

Abschließend noch ein Wort zur Digitalanzeige. Hier kann man wieder auf das Windows-System zurückgreifen. Sie ist ein Tochterfenster des Meßgeräts, das bei der Verarbeitung der `WM_CREATE`-Nachricht angelegt wird, bei `WM_DESTROY` automatisch wieder zerstört wird und der vordefinierten Klasse »static« angehört. Beim Bearbeiten von `WM_SIZE` wird es an die angezeigte Stelle geschoben und in der richtigen Größe abgebildet. Das Tochterfenster (der Enkel des Dialogfensters) wird nur dann immer richtig abgebildet, wenn im Fensterstil die Option `WS_CLIPCHILDREN` gesetzt wurde. Da man dies bei der Angabe innerhalb der Ressourcen-Datei leicht vergißt, wurde diese Option mit Hilfe von `SetWindowLong` bei `WM_CREATE` automatisch hinzugefügt.

Hinweise zur Entwicklung eigener Fensterklassen

Das Beispiel insgesamt zeigt deutlich, daß die Anwendung von eigenen Fensterklassen sehr einfach ist, die Implementierung dagegen ziemlich kompliziert. Bei letzterer handelt es sich um elementarste Grafikprogrammierung mit Berücksichtigung vieler Faktoren, von Ausnahmezuständen

```
    /* set count for long scale lines */
    r->uScaleDelta=(s<=Ø)? 32767:(r->sScale-1)/(s-1);
    if (r->uScaleDelta==Ø) r->uScaleDelta++;
    LocalUnlock(r->hmText);
    /* calculate scale locations */
    vfR1=Ø.9*(double)r->sxHalf; /* circle1: Ø.9 of 1/2 wid */
    vfR2=(double)r->sxHalf; /* circle 2: 1.Ø of half width */
    r->vfR3=1.1*(double)r->sxHalf; /* circle 3: 1.1 of 1/2 */
    vfR1Sqr=vfR1*vfR1; vfR2Sqr=vfR2*vfR2;
    r->vfR3Sqr=r->vfR3*r->vfR3;
    vfDelta=(9Ø.Ø/18Ø.Ø*PI)/(r->sScale-1);
    /* calculate scale points for right half */
    for (vfAngle=r->sScale&1? Ø.Ø:vfDelta/2,
          iTab1=(r->sScale-1)/2,iTab2=r->sScale/2;
          iTab1>=Ø;vfAngle+=vfDelta,iTab1--,iTab2++)
     {vfCos=cos(vfAngle);
      vfY1=vfCos*vfR1; vY1=(int)(r->vfR3-vfY1);
      vfY2=vfCos*vfR2; vY2=(int)(r->vfR3-vfY2);
      vfY3=vfCos*r->vfR3; vY3=(int)(r->vfR3-vfY3);
      vX1=(int)sqrt(fabs(vfR1Sqr-vfY1*vfY1));
      vX2=(int)sqrt(fabs(vfR2Sqr-vfY2*vfY2));
      vX3=(int)sqrt(fabs(r->vfR3Sqr-vfY3*vfY3));
      r->ScaleTable[iTab1].xStart=r->sxHalf-vX1;
      r->ScaleTable[iTab1].yStart=2*syChar+vY1;
      r->ScaleTable[iTab1].xStop=
         r->sxHalf-(iTab1%r->uScaleDelta==Ø? vX3:vX2);
      r->ScaleTable[iTab1].yStop=
         2*syChar+(iTab1%r->uScaleDelta==Ø? vY3:vY2);
      r->ScaleTable[iTab2].xStart=r->sxHalf+vX1;
      r->ScaleTable[iTab2].yStart=
         r->ScaleTable[iTab1].yStart;
      r->ScaleTable[iTab2].xStop=
         r->sxHalf+(iTab2%r->uScaleDelta==Ø? vX3:vX2);
      r->ScaleTable[iTab2].yStop=
         2*syChar+(iTab2%r->uScaleDelta==Ø? vY3:vY2);
     } /* for */
    /* set y origin of analog pointing line */
    r->yScaleCenter=2*syChar+(int)r->vfR3;
    /* set height of scale */
    r->syScale=r->ScaleTable[Ø].yStart+syChar;
    SetRect(&wrc,1,1,r->sxClient-2,r->syScale);
    InvalidateRect(hw,&wrc,TRUE); /* redraw complete scale */
    goto Return; /* return ØL */

  case WM_MOVE:
    /* set location of digital instrument */
    MoveWindow
     (r->hwDigital,(r->sxClient-6*sxChar)/2,
      r->syScale+(r->syClient-r->syScale-syChar)/2,
      6*sxChar,syChar+2,TRUE
     );
    break;

  case WM_PAINT:
    hdc=BeginPaint(hw,&wps);
    hpnDraw=CreatePen
     (PS_SOLID,1,GetSysColor(COLOR_WINDOWTEXT));
    hbrBkgr=CreateSolidBrush(GetSysColor(COLOR_WINDOW));
    SelectObject(hdc,hpnDraw); SelectObject(hdc,hbrBkgr);
    SetTextColor(hdc,GetSysColor(COLOR_WINDOWTEXT));
    SetBkMode(hdc,TRANSPARENT);
    Rectangle(hdc,Ø,Ø,r->sxClient,r->syClient);
    /* draw scale lines:
       use calculated values in r->ScaleTable */
    for (i=Ø;i<r->sScale;i++)
     {/* set start point */
      MoveTo
       (hdc,r->ScaleTable[i].xStart,r->ScaleTable[i].yStart);
      LineTo
       (hdc,r->ScaleTable[i].xStop,r->ScaleTable[i].yStop);
     } /* for */
    /* ----- draw scale text ----- */
    rzText=LocalLock(r->hmText);
    /* draw text of unity */
    rz=rzText;
    /* determine length of unity string */
    while (rz!=Ø&&*rz!='\t') rz++;
    SetTextAlign(hdc,TA_CENTER|TA_TOP);
```

Listing 8: *(Fortsetzung)*

```
   TextOut(hdc,r->sxHalf,(r->sxHalf/2+r->syScale)/2,
     rzText,rz-rzText);
   SetTextAlign(hdc,TA_CENTER|TA_BOTTOM);
   /* draw text at long scale lines */
   for (i=Ø;*rz;i+=r->uScaleDelta)
    {/* determine next line text: skip to end of part */
     rzText=++rz; while (*rz&&*rz!='\t') rz++;
     if (*rzText!='\t' && *rzText!='\Ø')
      {/* no empty text: draw it */
       TextOut
        (hdc,r->ScaleTable[i].xStop,
         r->ScaleTable[i].yStop,rzText,rz-rzText
        );
      } /* if */
    } /* for */
   LocalUnlock(r->hmText);
   /* ----- set gray cover rectangle ----- */
   SelectObject(hdc,GetStockObject(GRAY_BRUSH));
   Rectangle(hdc,Ø,r->syScale,r->sxClient,r->syClient);
   /* ----- draw line of analog pointer ----- */
   IntersectClipRect(hdc,Ø,Ø,r->sxClient,r->syScale);
   MoveTo(hdc,r->vxScale,r->vyScale);
   LineTo(hdc,r->sxHalf,r->yScaleCenter);
   EndPaint(hw,&wps);
   DeleteObject(hpnDraw); DeleteObject(hbrBkgr);
   break;
 case AM_UPDATE:
   /* ----- update line of analog pointer ----- */
   if (r->vyScale!=Ø)
    {/* remove old pointer line, invalidate scale range */
     hdc=GetDC(hw);
     hpnDraw=CreatePen
      (PS_SOLID,1,GetSysColor(COLOR_WINDOWTEXT));
     hbrBkgr=CreateSolidBrush(GetSysColor(COLOR_WINDOW));
     SelectObject(hdc,hpnDraw); SelectObject(hdc,hbrBkgr);
     SetROP2(hdc,R2_NOT); SetBkMode(hdc,TRANSPARENT);
     IntersectClipRect(hdc,Ø,Ø,r->sxClient,r->syScale);
     MoveTo(hdc,r->vxScale,r->vyScale);
     LineTo(hdc,r->sxHalf,r->yScaleCenter);
     ReleaseDC(hw,hdc);
     DeleteObject(hpnDraw); DeleteObject(hbrBkgr);
     SetRect
      (&wrc,min(r->vxScale,r->sxHalf),r->vyScale,
       max(r->vxScale,r->sxHalf)+1,r->syScale
      );
     InvalidateRect(hw,&wrc,FALSE);
    } /* if */
   /* set new analog pointer */
   vfAngle=PI/2*
    ((double)((int)uP1-r->uMinVal)/
    (double)((long)r->uMaxVal-(long)r->uMinVal)-Ø.5);
   /* overflow/underflow detection: reduce value */
   if (vfAngle<-1.2/4.Ø*PI)
    {vfAngle=-1.2/4.Ø*PI;
    }
   else if (vfAngle>1.2/4.Ø*PI)
    {vfAngle=1.2/4.Ø*PI;
    } /* if */
   vfY1=r->vfR3*cos(vfAngle);
   r->vxScale=
    r->sxHalf+(int)sqrt(fabs(r->vfR3Sqr-vfY1*vfY1))
    *(vfAngle<Ø.Ø?-1:+1);
   r->vyScale=2*syChar+(int)(r->vfR3-vfY1);
   SetRect
    (&wrc,min(r->vxScale,r->sxHalf),r->vyScale,
     max(r->vxScale,r->sxHalf)+1,r->syScale
    );
   InvalidateRect(hw,&wrc,FALSE); UpdateWindow(hw);
   /* ----- update digital value ----- */
   sprintf(z,"%d",uP1); SetWindowText(r->hwDigital,z);
   goto Return; /* return ØL */
 case WM_DESTROY:
   LocalFree(r->hmText); /* free allocated memory */
   /* destroy window data memory block, invalidate handle */
   LocalUnlock(hData); LocalFree(hData);
   SetWindowWord(hw,Ø,NULL);
   return ØL;
} /* switch */
```

Listing 8: *(Fortsetzung)*

```
  vRet=DefWindowProc(hw,iMsg,uP1,ulP2);
  /* return from function: unlock window data memory block */
  Return:
  if (hData) LocalUnlock(hData);
  return vRet;
 } /* fwAnalogMeter() */

/***********************************************************
 R e g i s t e r A n a l o g M e t e r
============================================================

This function registers the window class "RotationKnob".

Parameters:
 hi  is the handler of the module instance which possesses the
     window class.

Return:
 TRUE is returned if the window class was registered and FALSE
 if not.

***********************************************************/

BOOL RegisterAnalogMeter(HANDLE hi)

 {WNDCLASS wwc;

  wwc.lpszClassName="AnalogMeter";
  wwc.hInstance=hi;
  wwc.lpfnWndProc=fwAnalogMeter;
  wwc.hCursor=NULL; /* no selection => no cursor needed */
  wwc.hbrBackground=COLOR_WINDOW+1;
  wwc.style=CS_HREDRAW|CS_VREDRAW;
  wwc.hIcon=NULL; wwc.lpszMenuName=NULL;
  wwc.cbClsExtra=Ø; wwc.cbWndExtra=sizeof(HANDLE);
  /* register window, return status */
  return RegisterClass(&wwc);
 } /* CreateAnalogMeter() */
```

Listing 8: *(Ende)*

über Abbildungsbesonderheiten bis Performance-Untersuchungen und ergonomischen Gesichtspunkten. Daher ist es sinnvoll, sich bei der Planung eines Projekts frühzeitig zu überlegen, welche Objekte mit eigenen Klassen realisiert werden sollen, und diese so zu implementieren, daß sie leicht auf andere Projekte übertragbar sind.

Fensterklassen sollten immer sehr universell sein. Sie sollten sich der angegebenen Größe anpassen, entweder indem sie maßstäblich vergrößert werden (Rolleiste oder Drehknopf) oder bei konstanter Abbildungsgröße mehr Information darstellen (wie die Textfeld-Klasse »Edit«).

Wichtig ist, daß die Dialogobjekte unabhängig von der Bildschirmgröße oder -auflösung gleich groß sein sollten. Es ist empfehlenswert, wie bei der Funktion `CreateDialog` hierfür als Maßstab die Höhe und Breite eines Bildschirmzeichens zu verwenden. Oft besitzen Dialogobjekte verschiedene Dialog- und Zeichenvarianten, die jedoch nichts mit der Größe zu tun haben. Man kann diese entweder in Form von Nachrichten an die Objekte schicken (wie beim Analogmeßgerät), dies hat jedoch den Nachteil, daß man auf die Gestaltung eines Objekts nicht durch Änderung der Dialogfelddefinition in der Ressourcen-Datei Einfluß nehmen kann.

Es gibt zwei Möglichkeiten, Optionen oder Daten von der Ressourcen-Datei direkt an eine Fensterklasse zu übergeben. Die erste Möglichkeit ist der Fenstertext, dessen Dialogfeld-Angabe beim Anlegen des Felds automatisch an das Fenster geschickt wird. Dies wurde bei der Definition der Drehknopf-Skala beim Tongenerator gemacht. Die zweite Möglichkeit besteht darin, den Fensterstil zu verwenden. Auch auf ihn kann über die Ressourcen-Datei direkt Einfluß genommen werden. Er ist ein 32-Bit-Wert, von dem die oberen 16 Bit für Einstellungen von der Fensterverwaltung belegt sind (Tochterfenster, Rahmen, Rolleiste, Systemmenü etc.). Doch die unteren 16 Bit können für eine Fensterklasse privat genutzt werden. Dies ist insbesondere interessant, wenn eine Fensterklasse verschiedene Abbildungsvarianten besitzt. So realisiert die Fensterklasse »Button« sowohl Schaltflächen (*Push Button*) als auch eckige (*Check Box*) und runde (*Radio Button*) Optionsfelder, indem in den unteren 16 Bits für die einzelnen Dialogelemente verschiedene Werte zwischen 0 und 10 angegeben werden.

Alle Werte, die in Dialogfelddefinitionen verwendet werden können, können natürlich auch direkt in den Parametern der Funktion `CreateWindow` angegeben werden, sofern man damit Dialogobjekte außerhalb von Dialogfeldern anlegen muß.

Die Nachrichtenschnittstelle von Fensterklassen

Die Nachrichtenschnittstelle von eigenen Fensterklassen sollte sich im Aufbau immer nahe an den vordefinierten Fensterklassen von Windows orientieren, da es schon schwierig genug ist, sich in *diese* einzuarbeiten. In der Praxis bedeutet dies, daß nach Möglichkeit die Kommunikation mit den Fenstern einer eigenen Fensterklasse über bereits vorhandene Nachrichten von Windows erfolgt, wie beim Drehknopf die Verwendung der `WM_VSCROLL`-Nachricht. Ist dies nicht möglich, so können neue Nachrichten definiert werden, die folgenden Regeln genügen müssen:

- Nachrichten an Fenster einer neuen Fensterklasse können neu definiert werden im Bereich `WM_USER` (1024) bis 32767, sinnvollerweise beginnt man ab dem Wert `WM_USER` aufsteigend. Obwohl dadurch Nachrichten für verschiedene Fensterklassen den gleichen Wert besitzen, so beispielsweise `RK_GETRANGE` und `AM_UPDATE` den Wert (`WM_USER+1`), aber unterschiedliche Bedeutung haben, kann es dennoch zu keinen Verwechselungen kommen, da nie eine Nachricht `AM_UPDATE` an den Drehknopf oder `RK_GETRANGE` an die Analoganzeige geschickt wird. Die vereinbarten Nachrichten sind also immer einer Fensterklasse fest zugeordnet, deren Abkürzung sinnvollerweise den Namen der Nachricht einleitet (»RK_« oder »AM_«). Bei der Gestaltung dieser Nachrichten sollte man sich an den Standard-Windows-Nachrichten orientieren, die mit BM_, EM_ oder LB_ beginnen. Vergessen Sie nicht, daß viele Nachrichten auch einen Parameter zurückgeben können!

- Nachrichten, die ein Fenster an sein Vaterfenster sendet (sogenannte Mitteilungsnachrichten, *notification messages*), sollten nicht als neue eigenständige Nachrichten definiert werden, da sie mit Nachrichten anderer Fensterklassen kollidieren könnten. Hier empfiehlt es sich, dem von Microsoft eingeschlagenen Weg zu folgen und eine solche Nachricht als Argument einer `WM_COMMAND`-Nachricht zu übertragen, die dann folgendes Format in ihren Parametern besitzt:

Parameter	Bedeutung
`wParam`	enthält den Bezeichner des Dialogeintrags.
`LOWORD(lParam)`	enthält den Fensterbezug oder einen Parameter
`HIWORD(lParam)`	enthält den Wert der Mitteilungsnachricht, beginnend bei 0.

Oft übertragen Mitteilungsnachrichten keine weiteren Daten, sondern dienen lediglich dazu, ein bestimmtes Ereignis zu melden, beispielsweise, daß ein Eingabefenster angeklickt wurde oder ein Fehlerzustand eingetreten ist.

Bei der Gestaltung von Mitteilungsnachrichten eigener Fensterklassen sollte man sich an den Standard-Windows-Nachrichten orientieren, die mit BN_, EN_ oder LBN_ beginnen. Sind Mitteilungen über das Verschieben einer Rolleiste oder eines ähnlichen Positionier-Objekts zu machen, erfolgen diese über die Nachrichten `WM_HSCROLL` bzw. `WM_VSCROLL`.

Eine eigene Fensterklasse sollte folgende Standard-Nachrichten korrekt verarbeiten, sofern dies sinnvoll ist: `WM_ACTIVATE`, `WM_CREATE`, `WM_DESTROY`, `WM_ENABLE`, `WM_ERASEBKGND`, `WM_GETTEXT`, `WM_GETTEXTLENGTH`, `WM_KILLFOCUS`, `WM_MOVE`, `WM_PAINT`, `WM_SETREDRAW`, `WM_SETTEXT`, `WM_SETVISIBLE`, `WM_SHOWWINDOW` und `WM_SIZE`.

Falls es sich um eine Eingabe-Fensterklasse handelt, sollten zusätzlich folgende Standard-Nachrichten nach Möglichkeit bei der Implementierung berücksichtigt werden: `WM_CHAR`, `WM_CLEAR`, `WM_COPY`, `WM_CUT`, `WM_DEADCHAR`, `WM_GETDLGCODE`, `WM_GETFOCUS`, `WM_KEYDOWN`, `WM_KEYUP`, `WM_PASTE`, `WM_SETFOCUS`, `WM_xBUTTONDBLCLK`, `WM_xBUTTONDOWN`, `WM_xBUTTONUP` und `WM_MOUSEMOVE`.

Soll ein Dialogfenster einer eigenen Fensterklasse in unterschiedlichen Farben gezeichnet werden können, sollte die Rückfrage über die gewünschte Farbe an das Vaterfenster mit `WM_CTLCOLOR` erfolgen. Bedenken Sie immer, daß jemand Ihre Fensterklasse mit ungewöhnlicher Größe oder

Parametern aufrufen könnte. Wenn Sie Grenzen festlegen müssen, sollten diese klar dokumentiert sein. Fehler im Ablauf einer Fensterfunktion sollten über Mitteilungsfunktionen nach außen bekannt gemacht werden.

Bedenken Sie immer, daß jemand aus Ihrer Fensterklasse vielleicht eine Unterklasse machen möchte. Vermeiden Sie durch Einplanung solcher Möglichkeiten Fehler, wie sie sich beispielsweise bei der Implementierung der »Edit«-Klasse *[2]* eingeschlichen haben.

Es ist manchmal nicht ganz einfach, sich an alle angegebenen Regeln zu halten. Aber dadurch ersparen Sie dem Anwender eine aufwendige Einarbeitung in Ihre Fensterklasse und bewahren ihn vor unliebsamen Überraschungen bei der Anwendung.

Ein Ausblick in die Zukunft

Wie das vorgestellte Programm zeigt, kann man mit eigenen Fensterklassen das Aussehen eigener Applikationen unter Windows erheblich ändern, weniger, um wieder »alles ganz anders« zu machen als sich das Microsoft gedacht hat, sondern vielmehr, um die Benutzer-Ergonomie für die Eingabe oder Anzeige spezieller Daten erheblich zu verbessern. Eigene Fensterklassen erlauben eine saubere Trennung zwischen Implementierung und Einsatz von Dialog-Elementen. Die Implementierung solcher Elemente kann innerhalb eines Projekt-Teams unabhängig von einem Grafikspezialisten vorgenommen worden und innerhalb eines Unternehmens von einer spezialisierten Abteilung, die die Dialogobjekte einheitlich für eine ganze Produktlinie erstellt. Schließlich könnte man sich auch vorstellen, daß fertig gestaltete Dialogobjekte als Bibliothek zum Einbinden (ohne Quellcode) von unabhängigen Herstellern angeboten werden - dies als Hinweise für all diejenigen, die immer noch nicht glauben wollen, daß Windows und OS/2-Presentation-Manager die Standard-Dialogsysteme der Zukunft darstellen und gerade damit beschäftigt sind, die n-te Version eines eigenen Menügenerators zu schreiben oder gar die x-te Version eines ISAM-Treibers. Verlagern Sie ihre Kreativität in zukunftsträchtige und vor allem interessante Arbeitsgebiete!

Ich bin davon überzeugt, daß in einigen Jahren der Windows- oder OS/2-PM-Programmierer aus einer Fülle von fertig implementierten Dialogelementen wählen kann, die auf dem freien Markt erhältlich sind.

Marcellus Buchheit

[1] Microsoft Windows System Development-Kit, Version 2.

[2] Buchheit, Marcellus: Einführung in Fensterunterklassen, Microsoft System Journal, März/April 1989, Seite 67 bis 81.

[3] Welch, Kevin P.: Selbstdefinierte Dialog-Steuerungen, Microsoft System Journal, März/April 1989, Seite 58 bis 66.

[4] Petzold, Charles: Hexadezimaler Taschenrechner als Dialogbox, Microsoft System Journal, März/April 1988, Seite 24 bis 33.

Impressum

Das *Microsoft System Journal* erscheint alle zwei Monate (ungerade Monatszahlen) etwa Mitte des Vormonats.

Herausgeber, verantwortlich und Anschrift der Redaktion:
Microsoft GmbH, Redaktion *Microsoft System Journal*,
Erdinger Landstr. 2, D-8011 Aschheim-Dornach
Telefon: 089 / 46107-0, Teletex/Telex: (17) 89 83 28, Telefax: 90 63 55

Redaktion:
Günter Jürgensmeier, Haar und Hartmut Niemeier, Wildenberg

Mitarbeiter dieser Ausgabe:
Marcellus Buchheit, Greg Comeau, Günter Jürgensmeier, Hartmut Niemeier, Charles Petzold, Andrew Schulman, Richard Hale Shaw, Michael Tischer

Manuskripteinsendungen:
Manuskripte und Programmlistings werden von der Redaktion gerne angenommen. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck und zur Vervielfältigung der Programmlistings auf Datenträgern. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen. Nicht zur Veröffentlichung gelangte Manuskripte und Listings können nur zurückgeschickt werden, wenn Rückporto beiliegt.

Titelgestaltung: Hermann Menig
Anzeigenverkauf: Marianne Nuß
Druck und Abonnements:
schury praxisformulare GmbH, Postfach 270, D-8200 Rosenheim
Bezugspreise: Das Einzelheft kostet DM 19,80. Der Abonnementpreis beträgt DM 115,- für 6 Ausgaben und DM 210,- für 12 Ausgaben. Zu den einzelnen Ausgaben ist zum Preis von DM 19,80 eine Diskette mit allen Listings erhältlich. Das Abonnement inklusive Diskette kostet DM 230,- für 6 Ausgaben und DM 420,- für 12 Ausgaben. In den Preisen enthalten sind Mehrwertsteuer, Versandkosten und Zustellgebühren. Auslandsbezug auf Anfrage. Sollte die Zeitschrift aus Gründen, die nicht vom Herausgeber zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.
Bezugsmöglichkeiten: In Buchhandlungen und im Computer-Fachhandel. Abonnements und Einzelbestellungen: schury GmbH.

Das *Microsoft System Journal* wird mit Microsoft Word 4.0 geschrieben und gestaltet. Der Ausdruck erfolgt mit den HP-Softfonts AD und AF und dem Programm- und Schriftenpaket *DocuJet* auf einem HP-LaserJet Series II.



tewi Verlag GmbH · Theo-Prosel-Weg 1
8000 München 40 · Telefon 089/1292090

WINDOWS 2.0
Einführung + Referenz
Whitsitt/Bryan
496 S., DM 79,–
Auch als Einführung für WINDOWS/386 geeignet.

OS/2:
Einführung + Referenz
Beam, 400 S., DM 79,–
Kursartige Darstellung in 65 Modulen, zugleich als OS/2-Lexikon lesbar.

MS/PC DOS 4.0:
Einführung + Referenz
Stultz, 424 S., DM 69,–
Alle DOS-Befehle und Anwendungsfälle in 57 alphabetischen Befehlsmodulen.

C-Gesamtwerk
Herold/Unger
576 S., DM 79,–
zu Quick C, Turbo C, DR C, Lattice C, Intel C unter MS DOS, CP/M, UNIX, ISIS.

PC/XT/AT-Assembler-Buch
Thies, 650 S., DM 98,–
Für Assemblerprogrammierer und Systemsoftware-Entwickler. Speicherdiagramme. Anwenderbeispiele.

Der 80386
Thies, 496 S., DM 89,–
Einzigartige Erklärung des 80386-Konzepts, Architektur, Einsatz, Programmierung.

DRUCKFRISCH!



tm 5578/2

Für Eilbestellungen: Hotline 0 21 91-34 20 77